ON THE DESIGN AND EVALUATION OF RULE-BASED SYSTEMS

PRABHAKAR GOKUL CHANDER

A THESIS
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
CONCORDIA UNIVERSITY
MONTRÉAL, QUÉBEC, CANADA

JUNE 1996

ISBN 0-612-18425-0

Abstract

On the Design and Evaluation of Rule-based Systems

Prabhakar Gokul Chander, Ph.D.
Concordia University, 1996

In this thesis, we describe a three-tiered framework for the design, development and evaluation of rule-based systems. It consists of the following *stages*: the functional requirements stage, the design stage, and the implementation stage.

The functional requirements stage identifies the tasks that a system has to accomplish in its operating domain. At this stage, problem solving in a domain is viewed as a sequence of "goal-to-goal" progressions. Goals abstract the knowledge acquired from a domain. Two kinds of goals, intermediate and final, can be used to represent solutions to sub-problems and domain solutions, respectively. The set of goals and constraints specified for a domain constitute the *goal specification* of the domain. The problem solving in a domain is then modeled as a traversal of a *goal graph* of the domain.

The design stage sets certain restrictions for representing the specified goals in a rule base. However, the mapping between goals and their representation using hypotheses in a rule base is one-to-many. Thus, we can identify several *design schemes* for rule-based systems: the various design schemes impose different goal-to-hypothesis mapping restrictions. The design schemes are useful for both development and reverse engineering: for example, to improve the run time performance an existing rule base. The various design schemes and the relationships between them are studied in detail in this thesis. More specifically, we explore in detail a specific form of relationship between the design schemes, called *inheritance*. Its use and application are examined.

The implementation stage explicates the relationship between the rules in a rule base causing them to interact with each other while inferring goals. Goals are inferred in a rule base using partially ordered rule sequences called *paths*. Paths and goals can be used as a reliable basis for several evaluation procedures for rule-based systems. More specifically, we outline validation, performance and quality assessment, and verification for rule-based systems using paths and goals. The tools and algorithms developed to facilitate these evaluation processes are also described.

Acknowledgments

I would like to express my deep gratitude to my supervisors Dr. T. Radhakrishnan and Dr. R. Shinghal. Without their guidance, much of the research work reported in this thesis would not have been possible. I would like to particularly thank them for their patience and moral support. I have learnt much from their maturity and outlook. My research work was partly funded by a joint grant held by Dr. T. Radhakrishnan and Dr. C. Y. Suen. I thank them for their financial support.

I thank Alun Preece and Clifford Grossner. We spent time together in developing the initial version of the path model described in this thesis. I would also like to thank Alun for his help during the implementation of the Path Hunter tool described in this thesis. He also spent a significant amount of his time implementing the Path Tracer tool described in this thesis.

The Blackbox expert system mentioned in this thesis was designed and developed by Le Hoc Duong, John Lyons, and Clifford Grossner. Cliff provided the goal specification of the expert system required by Path Hunter.

An initial version of the Library expert system described in this thesis was developed by Dao Nguyen. The knowledge base of the expert system was developed after consulting Carol Coughlin and Lee Harris, reference librarians at Concordia University. Their help is gratefully acknowledged.

I thank the systems analyst group in the Computer Science department for their timely response and support whenever their help is needed. I would like to thank particularly Stan Swiercz, Michael Assels, Paul Gill, and William Wong for their help. In addition, the secretaries of this department have been very kind and helpful to me during the years of my study. I wish to convey my heartfelt gratitude to all of them.

Finally, I thank my parents and my sisters for their support during these years. I would particularly like to thank my mother Dr. (Mrs) Alamelu Prabhakar without whose encouragement, I would not have decided to do my doctoral degree. In appreciation, I dedicate this thesis to my parents.

Contents

List of Figures

Chapter 1

Introduction

Broad issues that confront developers and researchers alike in the development of rule-based systems are introduced. The related research work is briefly surveyed to provide motivation to the approach in this thesis.

1.1 Building Rule-based Systems

In the construction of any software system, system builders adopting an engineering approach have traditionally emphasized the need for a disciplined manner in which the system is specified, designed, implemented, and evaluated [Ghezzi, Jazayeri, and Mandrioli, 1991]. Specification is a precise statement of the requirements of the problem that the system is to solve. The design of a software system can be viewed as applying successive refinements to the specification of a problem, transforming it stage by stage, until it can be implemented on a computer. Stated alternatively, the design phase maps the problem into a solution model, and the implementation phase represents this solution model so that it can be executed directly by a computer. The evaluation phase ensures that the system meets the need for which it is constructed.

Rule-based systems are a special type of software systems that are developed using a declarative programming paradigm [Shinghal, 1992; Giarratano and Riley, 1993]. The applications of rule-based systems often differ from that of traditional software systems. For example, rule-based systems are typically used to solve diagnosis type of problems, such as diagnosing diseases from a given set of symptoms, rather than solving problems such as matrix multiplication, or matrix inversion [Shinghal, 1992; Lucas, 1994].

Formally, a problem P can be described using a set of ground atoms in first order logic denoting a permissible combination of initial evidence [Shinghal, 1992], where an atom consists of a predicate with its arguments. Each problem P is associated with at least one solution. Given a problem P, if a system finds its solution, then the system is said to have solved the problem. The set of problems that can be input to a system for solutions is said to constitute the **problem domain** (or, simply, domain) of the system. However, a system may not be able to solve all the problems in a domain. The set of solutions associated with the problems in a domain constitutes the **solution range** (or, simply, range). The notions of problem, problem domain, and problem solving are portrayed in Figure 1.1. As an example, consider identifying the occupation of persons in a university domain. The problems in the domain consists of sets of initial evidence describing the characteristics of persons working in the

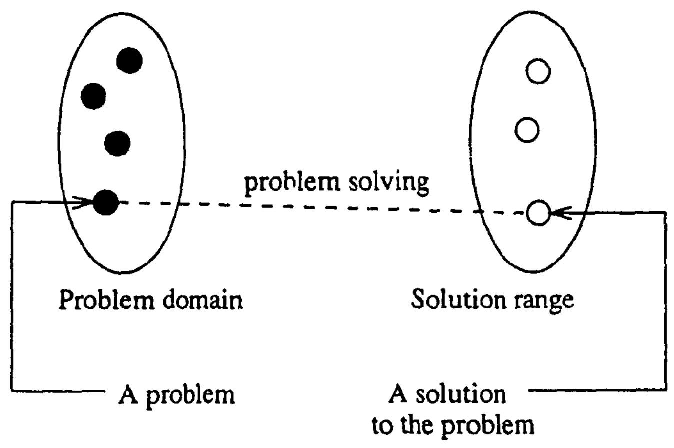


Figure 1.1: *Illustrating problem, problem domain, and solution.*

university, and their solution consists of the roles of the persons in the university.

The process of acquiring the knowledge required to solve problems in a given domain (say, medicine) constitutes **knowledge acquisition** [Yost, 1993; Krause, Fox, Neil, and Glowinski, 1993]. While there are several techniques that have been developed, the most popular one is that of interviewing an expert in the domain to obtain his/her *mental view*: how he/she would typically solve a given problem, for example, diagnosing a disease from a given set of symptoms.

Once the problem solving knowledge is acquired for a domain, it can be encoded to develop a rule-based system for the domain. A rule-based system consists of the three following components:

- A **rule base** that contains a set of declarative constructs called rules. A rule is akin to a statement in a procedural programming language in the sense that it is an unit of execution.

- An execution mechanism called the **inference engine** that controls rule execution. The execution of a rule is called *rule firing*, and the computation required to check if a rule can fire is called *pattern matching* in the literature [Shinghal, 1992; Giarratano and Riley, 1993]. However, unlike a procedural programming language, the order in which a set of rules are fired need not be the same as their physical order in the rule base. It is determined by the *inference strategy* (a criterion to order the rule firings in a certain way) used by the inference engine.

Two of the common inference strategies are as follows: cataloged-order (a rule r_i fires before r_{i+1}, if r_i is cataloged before r_{i+1} in the rule base), priority based (rule firings are ordered by a rule priority usually assigned by the user) [Shinghal, 1992]. For a given inference strategy it is possible to enumerate the rule firing sequence, however, for an arbitrary inference strategy no assumption can be made on the order of rule firings [Giarratano and Riley, 1993].

- A dynamic data base for holding data produced as a result of rule executions called the **working memory**.

Problem solving in a rule-based system occurs as follows: the inference engine chooses a set of rules to fire for a given input. The firing of this set of rules produces data that *causes* other rules to fire. Typically, the rule firings continue until no more rules can fire, or until some solutions are inferred. The set of rules fired for a given input constitutes the **run time behavior** (or, simply, behavior) of the system.

During problem solving, a given rule can fire at different times and can cause several other rules to fire. The term **rule interaction** is frequently used in the literature to indicate the effect of this causality relationship between the rules during problem solving [Giarratano and Riley, 1993; O'Neal and Edwards Jr., 1993]. Owing to the non-sequential nature of rule executions in a rule-based system, the analysis of the interactions between the rules to understand its behavior better is an actively pursued topic [Chang, Combs, and Stachowitz, 1990; O'Neal and Edwards Jr., 1993; Kiper, 1992].

To summarize, there are two key characteristics of rule-based systems that distinguish them from a traditional program, making them harder to understand, evaluate, and maintain:

- the declarative nature of knowledge and the control representation in rules that characterize them as non-sequentially executing systems [Clancey, 1983; Shinghal, 1992]; and
- the interactions that occur between the rules [O'Neal and Edwards Jr., 1993].

As a software system, the design and development of a rule-based system follows a pre-defined set of phases and processes that characterizes its **life-cycle** [Ghezzi et al., 1991; Batarekh, Preece, Bennett, and Grogono, 1991; Lee and O'Keefe, 1994]. Thus, in developing a rule-based system for a domain, there are several criteria that should be considered [Liebowitz, 1989]. But, strict sequential flow models of development such as the waterfall model [Boehm, 1977] used for traditional software engineering projects are not applicable for the development cycle of rule-based systems primarily because of the following reasons:

- their operating domains are said to be "ill structured" in nature because a precise specification of the domain's characteristics may not be known [Simon, 1973; Krause et al., 1993]: for example in the domain of medicine, it is not possible to precisely specify the problem of disease diagnosis;
- the problems that are confronted by these systems often do not have a clear algorithmic solution [Shinghal, 1992; Pople, 1982]; and
- the domain knowledge evolves with time.

Even though several life-cycle models have been proposed for rule-based systems, a gradual evolutionary process of each phase in the life-cycle has been pointed out [Weitzel and Kerchberg, 1989; Batarekh et al., 1991; Lee and O'Keefe, 1994]. The evolutionary process causes refinement of not only the partially developed system, but also its requirements [Yost, 1993]. More precisely, the life-cycle of a rule-based system tends to follow a spiral model of evolutionary development [Boehm, 1988; Giarratano and Riley, 1993; Lee and O'Keefe, 1994]. Thus, accommodating incremental development should be part of any design methodology for rule-based systems.

1.2 Design, Evaluation, and Maintenance of Rule-based Systems

Researchers have been working on the design and development frameworks, and evaluation methodologies for rule-based systems for the past several years [Chandrasekaran, 1986; Liebowitz and De Salvo, 1989; Debenham, 1992; Hamilton, Kelley, and Culbert, 1991; Gupta, 1993; Schreiber, Wielinga, and de Hoog, 1994]. The current approaches to the construction of rule-based systems focus on the automation of knowledge acquisition and rule base generation, and on methods for evaluating and maintaining large rule bases. To facilitate automated rule base generation, the structure of the problem to be solved is described using an appropriate problem solving model and a language [Yost, 1993; Marcus and McDermott, 1989]. A common approach for evaluating a rule-based system is based upon detecting anomalies in its rule base, and by measuring the extent rule sequences in a rule base are exercised for various test cases [Gupta, 1993; Preece, 1992; Zlatareva and Preece, 1994]. To facilitate the maintenance of large rule-bases, researchers provide design restrictions, separate the control knowledge from the domain knowledge, modularize the system by having several rule-bases, or provide rule groupings to localize a change to a subset of rules [Clancey, 1983; Jacob and Froscher, 1990; Debenham, 1992; Antoniou and Wachsmuth, 1994].

1.2.1 Designing Rule-based Systems

Four representative approaches described in the literature for designing rule-based systems are presented below. More specifically, the Generic Tasks approach, the SALT approach, the SOAR framework, and the KADS methodology for rule-based systems are discussed.

Generic Tasks A set of six generic tasks, general enough to include a wide range of existing rule-based systems, are identified in order to facilitate building and understanding knowledge based systems [Chandrasekaran, 1986; Bylander and Mittal, 1986]. These tasks are as follows: (1) hierarchical classification (applicable for diagnosis type of problems); (2) hypothesis matching; (3) knowledge directed information passing (that pertains to general domain knowledge such as domain attributes); (4) abductive assembly (used to assemble a set of hypothesis inferred into a single "best" hypothesis to facilitate explanation); (5) hierarchical design (by plan selection and refinement); and (6) state abstraction (to predict the consequences of an action). An integration of some of these generic tasks is used to represent the structure of problems in a domain. The underlying philosophy of this approach can be summarized as follows: instead of representing all types of knowledge about different domains using the same rule language (of a rule-based system shell) and inferring hypotheses in a uniform way, the domain knowledge representation should be made in a specific language that best reflects the task structure.

The SALT Approach SALT is a knowledge acquisition language for propose-and-revise systems, and a tool associated with this language enables the generation of rule-based systems [Marcus and McDermott, 1989]. SALT is intended for use by domain experts, and is meant for the creation and maintenance of systems that are suited to constraint satisfaction tasks such as scheduling. Domain knowledge is represented through a set of design parameters, their values, constraints associated with the values, and a set of constraint fixes. SALT constructs a dependency network of these parameters, performs some consistency checks, and applies fixes whenever constraints are violated. Once the domain specific knowledge is represented in a dependency network, SALT can generate a set of OPS5 rules [Forgy, 1981] .

The SOAR Framework The philosophy behind this approach is to avoid the domain (or task) specific disadvantages of method based tools like SALT [Yost and Newell, 1989; Yost, 1993]. SOAR is based on a computational model that is general enough to represent a variety of problem structures, called Problem Space Computational Model (PSCM). A computer representation of PSCM is provided using a language called TAQL (Task Acquisition Language). The components of PSCM are tasks and problem spaces; problem spaces are comprised of states, operators and the knowledge associated with the application of these operators. SOAR is an environment for PSCM, and provides an execution platform for the rules that implement the solution of a problem expressed in PSCM. The TAQL compiler generates SOAR rules once the problem modeled in PSCM is represented using TAQL.

The KADS Methodology The goal of this methodology is to provide a set of generic re-usable knowledge parts, or components that can be used by developers in system construction. In KADS methodology, development is essentially seen as a modeling activity [Aben, 1993; Schreiber et al., 1994]. Specification and design are viewed as construction and refinement of models describing problem solving in the domain. In this process, the model of expertise (typically obtained from a domain expert) in problem solving is divided into four layers: the *domain* layer describes static and axiomatic knowledge (domain terms); the *inference* layer is responsible for the roles of the various knowledge entities and what new information can be made from existing information; the *task* layer is concerned with task (problem) decomposition and relating it to the inference layer; and the *strategy* layer specifies what goals need to be reached to solve problems. The inference layer is responsible for problem solving [Aben, 1993], where a typical problem solving scenario takes place is as follows: the strategy layer chooses a set of tasks to be solved from the task layer, and they are solved by the inference layer using the knowledge description in the domain layer.

In the Generic tasks and SOAR approaches, the focus is on the design and development, but evaluation and maintenance issues are not explicitly addressed. The Generic tasks approach has been criticized to have limited re-usability as the granularity of a generic task is rather large [Aben, 1993], and the automated rule base generation using TAQL in SOAR [Yost, 1993] has its applicability limited by the effectiveness with which a domain expert can communicate his task knowledge, and its description in TAQL. The SALT approach provides a method for functional testing, but lacks general applicability, and has been criticized as a "domain (task) specific approach" [Yost and Newell, 1989].

The KADS methodology is elaborate in its description of system design, and development [Schreiber et al., 1994], but its models are too complex and its practical use is yet to be seen. In fact, the KADS ontology is criticized as incomplete, and names and labels of the inference steps are often ambiguous or confusing [Aben, 1993]. For example, not all proposed inference steps are at the same level of complexity; the inference steps lack clear description; and sometimes the inference description is too general for specific tasks. In addition, many KADS variations that are available [Schreiber et al., 1994; Wells, 1993] which can confuse developers as to which indeed is the real KADS methodology. To make use of this methodology, the knowledge engineers must have a library of modeling components, but such a generic "reusable" library can only have a set of components that will require some "fine tuning" for the underlying domain [Aben, 1993]. Although, a separate generalized inference layer can facilitate maintenance, the details of evaluation and issues in maintenance in this framework are not explicitly addressed [Schreiber et al., 1994]. For practical purposes, a design framework that makes use of the "structure-preserving" philosophy of KADS (preserving the link between the knowledge model and the actual implementation) would be pragmatic and useful.

1.2.2 Evaluating Rule-based Systems

Evaluation is a process of ascertaining that a system indeed meets the need for which it is constructed [Ghezzi et al., 1991]. The evaluation processes for rule-based systems have been classified into three major groups: verification for (static) detection of rule base anomalies [Ginsberg, 1988; Rousset, 1988; Preece, Shinghal, and Batarekh, 1992b; O'Keefe and O'Leary, 1993], validation for uncovering (dynamic) behavioral errors [Chang et al., 1990; Kiper, 1992; Preece and Shinghal, 1992; Preece, Grossner, Gokulchander, and Radhakrishnan, 1995], and performance assessment to measure other qualities such as efficiency, reliability, etc [Preece, Grossner, Gokulchander, and Radhakrishnan, 1994; Giovanni, 1989; Guida and Mauri, 1993].

Rule bases can inadvertently suffer from four anomalies abbreviated in the literature as CARD [Preece et al., 1992b; Preece, Shinghal, and Batarekh, 1992a]: Circularity, Ambivalence, Redundancy, and Deficiency. Early verification procedures to detect these anomalies had limited anomaly detection capability [Suwa, Scott, and Shortliffe, 1982]. Other methods provide some computational advantage over [Suwa et al., 1982] by using a pairwise rule comparison method to detect anomalies [Cragun and Steudel, 1987; Nguyen, Perkins, Laffey, and Pecora, 1985], but are criticized as incomplete as they can miss detecting some anomalies [Ginsberg, 1988]. Some approaches make use of the meta knowledge of the domain for a more detailed anomaly detection: for example, the structure checker in EVA [Chang et al., 1990] makes use of atom synonyms to detect redundancy. However, the method of anomaly detection favored by most researchers is one that takes into account the transitive inference chains in a rule base [Ginsberg, 1988; Rousset, 1988], but these methods are computationally expensive, or have limited applicability. In some works, a rule base is modeled as a graph and graph properties are used to detect anomalies, but the algorithms are often complex, or cannot be extended for large rule bases [Suh and Murray, 1994; Valiente, 1993]. The use of petri-nets and their properties to model a rule base [Agrawal and Tanniru, 1992; Nazareth, 1993]; use of machine learning approaches [Lounis, 1993; Meseguer, 1993]; modeling a rule base as a set of in-equations [Prakash, Subramanian, and Mahabala, 1991]; use of formal proof techniques [Waldinger and Stickel, 1991]; and using a feedback control system to model a rule base [Lunardhi and Passino, 1991], etc, have been attempted by researchers for anomaly detection. But, these approaches suffer from one or more of the following problems: scalability, computational intractability, and limited scope of applicability. As a final note, verification when extended to hybrid systems—systems using both declarative and frame based methods for knowledge representation—takes a deeper connotation and anomaly detection is a much harder problem [Lee and O'Keefe, 1993; Mukherjee and Gamble, 1995].

The functional aspect of validation ensures correct working of a system, whereas the structural aspect ensures that the structural sub-components of a system also perform correctly, and contribute to its correct behavior. Most of the existing validation methods are either functional, or empirical in nature. In a functional approach, the system solutions are compared with pre-solved solutions to often random test cases, whereas in an empirical approach they are compared with the solutions of a domain expert [Vinze, 1992; Preece, 1992; Zualkernan and Lin, 1993; McDuffie, Smith, and Flory, 1994]. But, random functional testing is often criticized to be inadequate [Plant, 1992], and there is a need for a systematic design of test cases and automated test case generation tools [Preece and Shinghal, 1992; Vignollet and Lelouche, 1993]. For rule-based systems, generating a representative set of test cases to test the system adequately is difficult; the test cases are often biased [Chang et al., 1990; Preece and Shinghal, 1992; Preece, 1992]; and they do not exercise a significant portion of the rule base [O'Keefe, Balci, and Smith, 1987]. Thus, a structure-based validation consisting of clearly identifying the sub-components of a system, generating test cases based upon this identification, and exercising as many of these sub-components as possible [Rushby, 1988; Preece, 1992] would be a reliable way to judge its acceptability. To further improve the effectiveness of such a validation procedure, some researchers advocate use of "meta knowledge" of the domain [Plant, 1993].

However, most developers often refrain from performing structural validation as it has been perceived to be "difficult" [Hamilton et al., 1991]. The difficulty in structural validation for rule-based systems lies in the formulation of structural components of the system so that they can be tested. The most common approach is to define this notion as rule sequences in the rule base, called paths of a rule base, thus accounting for the transitive inference chains and the rule interactions. By measuring the extent these rule sequences are exercised by a selected set of test cases, one can obtain a better insight about the system behavior. The individual structural validation methods differ in the way they define a rule base path [Rushby, 1988; Chang et al., 1990; Kiper, 1992], but suffer from computational intractability, ambiguity, and/or inaccuracy [Preece, 1992].

Performance and quality assessment are evaluation techniques concerned with certain system qualities such as efficiency, adequacy, reliability, etc. Approaches for performance and quality assessment of rule-based system provide methodologies [Guida and Mauri, 1993], make specific measurements focusing on one quality (often reliability) [Giovanni, 1989], or extend metrics used in software engineering for rule-based systems to quantify some of their qualities [O'Neal and Edwards Jr., 1993; Chen and Suen, 1993; Kiper, 1992].

The current approach towards rule-based system evaluation is based upon the notion of integrating the evaluation processes as part of its life-cycle rather than viewing them as a post-development phase [Wells, 1993; Andert Jr, 1993; Lee and O'Keefe, 1994]. These approaches emphasize that the evaluation processes should start from knowledge acquisition itself [Long and Neale, 1993; Mengshiel, 1993] for the following reasons: (1) the requirements of rule-based systems cannot be specified formally [Krause et al., 1993]; (2) system developers are often confused regarding 'when and how' to perform system evaluation [Andert Jr. 1993; Hamilton et al., 1991]; and (3) improving the quality and reliability of rule-based systems is non-trivial. Hence, by ensuring that evaluation spans all the phases of a life-cycle, it is hoped that the quality and reliability of rule-based systems can be improved.

1.2.3 Maintaining Rule-based Systems

A system that is developed and delivered for operation in a field should be maintained thereafter until it is de-commissioned from service. Maintenance is a process of enhancing the system to changing requirements as well as fixing errors encountered that were not detected earlier, and can incur as much as 60% of the total cost associated with the system [Ghezzi et al., 1991]. Further, the person who maintains the system need not be the one who developed the system; hence, the choices made during system design to improve its understandability play a major role in maintenance [Clancey, 1983; Jacob and Froscher, 1990; Yen, Juang, and MacGregor, 1991; Antoniou and Wachsmuth, 1994]. However, most of the existing approaches tackle maintenance from a view point of modifications, and often ignore processes other than maintenance that are part of a system's life-cycle. For example, the issues of incremental evaluation of a rule base to conserve evaluation costs [Meseguer, 1992], and the impact of changing existing design decision(s) during maintenance are often not explicitly addressed.

To facilitate the maintenance of large rule-bases, researchers attempt to separate the control knowledge from the domain knowledge and encode the control (or inference) knowledge in the form of metarules to improve the understandability of its design [Clancey, 1983].

A more common approach for rule-based system maintenance is based upon rule groupings [Jacob and Froscher, 1990; Mehrotra, 1993, 1995]. To reduce the effort associated with rule base modifications, it should be localized to a smaller set of rules; this set of rules is called a rule group [Jacob and Froscher, 1990]. Each group of rules should ideally encode a method for solving a particular task. Once the rule grouping is complete, the group dependency is documented by means of the facts that connect one rule group to another, called inter-group facts; thus, it is expected that a change can often be localized to a given rule group. By preserving the correct functioning of the rules in the group, and the assertions relating to inter-group facts, the knowledge engineer can make a given change with ease, and can be confident that the change does not affect the overall correctness of the system. In [Mehrotra, 1993, 1995], a similar rule grouping method called multi-viewpoint clustering is adopted to improve the understandability of the system. Though the clus' ing mechanism is meant for facilitating evaluation, its goal is to *understand* the knowledge encoding in a rule base; this is crucial if changes need to be made on the existing rule base. The method is based upon grouping related rules, called clusters, based on a distance measure between the rules that measures the extent of their cohesiveness (the extent how various items in the rules are related), and coupling (based on the properties shared by rule groups). By such rule groupings, changes can be identified into a localized setting. In [Vestli, NordbØ, and SØlvberg, 1994], the authors describe a tool that supports rule grouping and controls their execution, and its use in design and maintenance. A variation of rule groupings is rule base modularization: having several small rule bases called a module, with clearly documented module interfaces to improve their maintainability [Antoniou and Wachsmuth, 1994].

Debenham (1992) approaches the issue of maintenance from a design point of view. By imposing certain constraints in the knowledge to be encoded as part of system design, it is proposed that the maintainability of the system can be improved. The design methodology consists of an informal application model that documents the application, and three formal system models to describe the data, data relationships and constraints. By a process of normalization, the link between every data item d and other data items related to d (that could be affected by a modification to d) is kept. Thus, for maintenance, modification of an item requires locating the item followed by a check on those items that are linked to this item. The normalization process, however, can be computationally quite expensive, and though the approach only partially automates maintenance operations, even such partial automation is valuable.

The use of object oriented methods to improve maintainability and re-usability of large rule bases is explored in [Yen et al., 1991]. A rule's consequent in this method corresponds to a generic function that can activate several methods, which in turn can cause other rules to fire. Thus, the implementation of a rule's function is hidden. Any changes to a rule's function should, ideally, require changing only the implementation. By automatically keeping related rules in groups based upon their function, the authors claim the understandability (hence, maintainability), and re-usability can be improved. The practical use of this approach, however, is yet to be seen.

1.3 Approach of this Thesis

This thesis is concerned with the design and evaluation of rule-based systems. It is aimed at answering the following questions as part of a design framework proposed for rule-based systems.

(a) what characteristics of the "design stage" for rule-based systems can serve as useful guidelines for developers?

(b) how do we map a design choice to its corresponding implementation?

(c) how do we ensure that the implementation and its associated design choice indeed represent a given requirement identified from the acquired knowledge?

(d) how do we evaluate the developed rule base?

A detailed study of the literature shows a need for a design framework that is easy for developers to adopt, but comprehensive enough to handle the above issues.

An analysis of the above development issues reveal that we need a "link" between the various phases of a life-cycle, if we are to map a part of one phase to that of the preceding phase. Note, however, deciding on this link is non-trivial: the decision of what constitutes such a link can impact not only the evaluation processes, but also maintenance [Debenham, 1992; Wells, 1993].

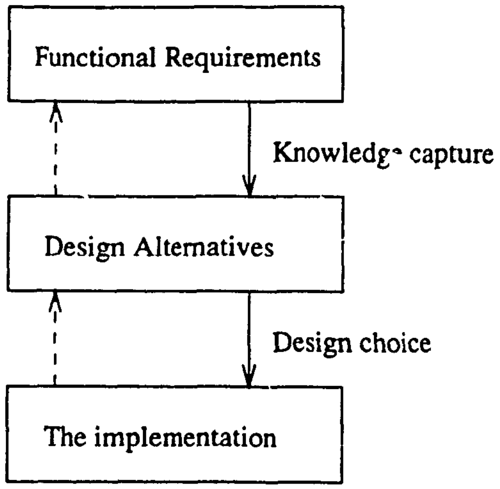


Figure 1.2: *A software development perspective for rule-based systems.*

The proposed design framework adopts a slightly modified linear life-cycle model[1] of rule-based systems assumed embedded on a spiral model of development to accommodate incremental evolutions [Giarratano and Riley, 1993]. This is shown in Figure 1.2. Each rectangular box in the figure represents a phase in the life-cycle; more precisely, the execution of the activities appropriate to a rectangular box is a phase. For simplicity, each of the boxes are referred to as **stages**. The dotted lines in the figure imply that the reverse links shown between the stages, mapping system implementation back to its conception, are not explicit. The details of these stages are described in the subsequent chapters. A brief description appears below.

The functional requirements of a rule-based system are captured using a reference to abstract the problem solving knowledge. The design stage formalizes the acquired knowledge concepts, and sets certain restrictions for the implementation of the rule base. The implementation stage uses a model to capture rule interactions occurring at the rule base level so that problem solving that takes place by rule firings can be mapped to identify definite portions of the knowledge acquired in the functional requirements stage. This model can then be used as a reliable basis for evaluating rule-based systems. System evaluation in this frame work can thus be considered integrated with all the stages of the life-cycle.

[1]In the software engineering literature, this model is also called "waterfall model with feedbacks" [Ghezzi et al., 1991].

1.4 Organization of the Thesis

The thesis is organized as follows. The next chapter (chapter 2) describes the details of identifying the functional requirements of a rule-based system by using an abstraction of the acquired knowledge. The proposed approach is based upon viewing problem solving as "goal-to-goal" progressions; thus, we identify and specify goals to be achieved and other constraints to be satisfied as part of the functional requirements of a system. A careful identification and specification of goals can be used for optimizing the performance of an existing rule base. This is described using a case study.

Once goals are identified, the next step is to identify the properties that should be satisfied in order to represent a goal in a rule base. More specifically, chapter 3 examines the restrictions that should be satisfied by a goal at the implementation stage. This gives rise to several "design schemes" for rule-based systems. Their relationships are analyzed to provide insights into system development. The case study described in chapter 2 is re-designed to exhibit the practicability of this approach for development. reverse engineering and development.

The implementation of a rule-based system for problem solving should meet its functional requirements. Chapter 4 describes a model of a rule base to understand the problem solving that would take place in a system. This is based upon identifying how rule interactions cause goals to be inferred during problem solving. The use of this model for system evaluation is then described. Finally, chapter 5 summarizes the work done, its contributions, and outlines avenues for future research.

Chapter 2

Goal Specification of a Domain

Goals are used to abstract the knowledge acquired from a domain. Two kinds of goals, intermediate and final, can be used to indicate solutions to sub-problems and domain solutions respectively. The problem solving in a domain can then be modeled as a graph traversal. Goals are also useful for restructuring an existing rule base to improve its performance. Rule base restructuring is described using a case study.

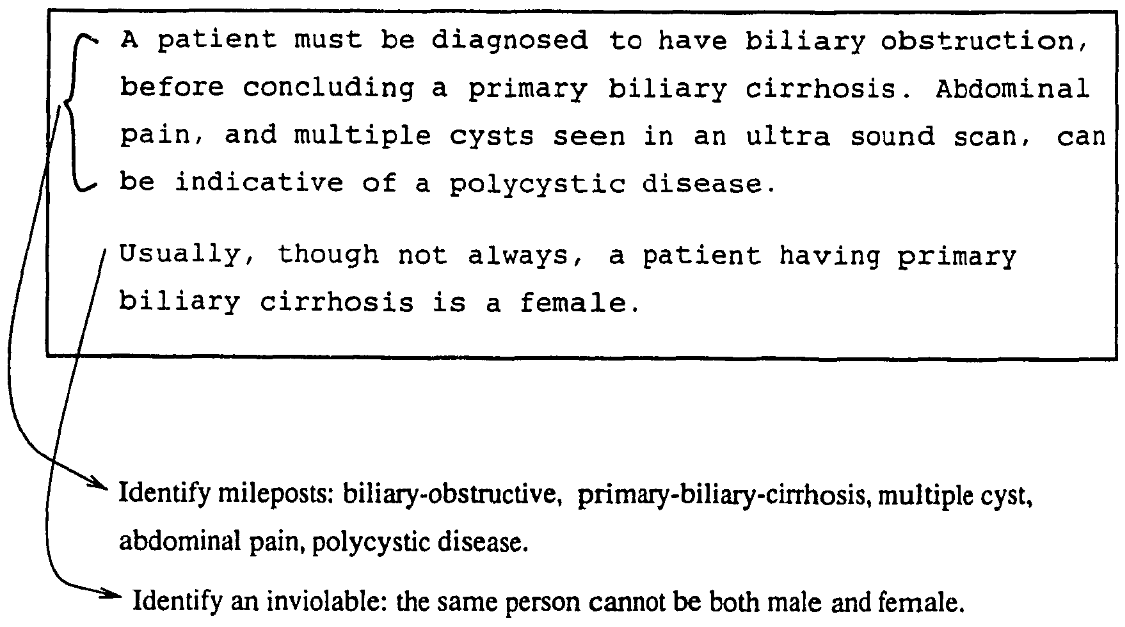
2.1 Abstracting Domain Knowledge by Goals

Formal and rigorous approaches to knowledge acquisition for rule-based system development are currently favored by researchers [Yost and Newell, 1989; Batarekh et al., 1991; Krause et al., 1993] because they are believed to improve the quality and reliability of a system. In this approach, problem solving in a domain is viewed in the form of state progressions as in traditional AI research [Rich, 1991; Shinghal, 1992]. In order to solve a problem, a system will transit through a set of states. It is unrealistic to enumerate every possible state that is traversed by the system without some sort of abstraction over the state space. A domain expert solving problems in the domain knows of the typical mileposts that need to be accomplished as part of solving problems in the domain; referred to as **goals**, such mileposts can be specified.

Goals are abstractions of states, each representing some concepts of a domain obtained during knowledge acquisition. For example, in a medical diagnosis domain, inferring a `biliary obstruction disorder` is necessary towards inferring a final diagnosis `primary biliary cirrhosis` [Lucas, 1994]. Thus, the concept of `biliary obstruction` represents an abstraction of a state that serves as a milepost before inferring `biliary cirrhosis`, which is another state abstraction of a more specific concept (in this case, a solution).

The process of identifying and mapping the acquired knowledge into goals is not a mechanical process [Yost and Newell, 1989]. The extent to which a domain expert communicates the knowledge clearly and unambiguously, the skill of the knowledge engineer to abstract the body of knowledge acquired, and the rigor of the knowledge acquisition process, all play major roles in identifying goals from a given knowledge description. In this sense, goal identification is similar to the way problem concepts are identified and mapped to operators and states in SOAR [Yost, 1993].

Every goal, when translated into a first order logic formula, consists of a conjunction of hypotheses, where each hypothesis is represented as an atom. The atoms that are present in goals are called **goal atoms**; the other atoms are **non-goal atoms**, they being needed for rule encoding. In addition, the domain expert also specifies



A patient must be diagnosed to have biliary obstruction, before concluding a primary biliary cirrhosis. Abdominal pain, and multiple cysts seen in an ultra sound scan, can be indicative of a polycystic disease.

Usually, though not always, a patient having primary biliary cirrhosis is a female.

Identify mileposts: biliary-obstructive, primary-biliary-cirrhosis, multiple cyst, abdominal pain, polycystic disease.

Identify an inviolable: the same person cannot be both male and female.

Figure 2.1: *The functional requirements stage where the acquired knowledge is mapped to identify goals, and inviolables. Note, the goals at this stage only abstract the acquired knowledge. The decision of how to represent each goal in the system would be a design decision.*

constraints, called **inviolables**, associated with the domain; an inviolable is a conjunction of hypotheses such that all of them should not be true at the same time. An example of an inviolable is $MALE(x) \wedge PREGNANT(x)$; it is obvious that no goal should contain an inviolable. To illustrate, Figure 2.1 shows an example knowledge description, identifying goals and inviolables, and Figure 2.2 shows the translation of some of the goals identified in Figure 2.1 into a conjunction of atoms denoting hypotheses.

Goal Identified	Represented concept	Goal Translation
Biliary cirrhosis	A diagnosis representing primary biliary cirrhosis.	P-BILL-CIRR($x, sex, case\_hist$)
Multiple liver cysts	An intermediate diagnosis indicating cysts in liver.	CYSTS($x, sex, diag\_tchnq, prev\_hist$)
Abdominal pain	A symptom associated with polycystic disease.	PART($x, sex, desc$) $\wedge$ SYMPTOM(s_desc)

Figure 2.2: *Translating some of the goals identified in Figure 2.1 into a conjunction of first order logic atoms.*

The set of goals and inviolables identified for a domain constitutes the **goal specification** for the domain [Chander, Shinghal, and Radhakrishnan, 1994, 1995b, 1995c; Chander, Radhakrishnan, and Shinghal, 1995a]. Goals also serve as "meta knowledge" of the domain that would be useful later for verification and validation (chapter 4, sections 4.3, 4.4, and 4.5) The use of goals for system evaluation is discussed in chapter 4.

2.2 Modeling Problem Solving Using Goals

While solving problems, two kinds of goals are inferred: goals that facilitate inferring a solution, and the goals that represent the solutions to problems in the domain. The former are called **intermediate** goals and the latter **final goals**. For the example shown in Figure 2.1, `multiple cyst` is an intermediate goal and `primary biliary cirrhosis` is a final goal. Typically, the intermediate goals are those that are achieved in order to infer a final goal, and a final goal is part of some solution. It should be noted that there can be many solutions to a given problem.

Definition 1 (Intermediate and Final Goals) *Goals that are inferred in order to facilitate reaching a solution are called* intermediate goals. *The goals that are used for indicating domain solutions are called* final goals.

Solving a problem in terms of goals can be viewed as a succession of goal inferences starting from a permissible combination of initial evidence to inferring a goal, and then moving from goal to goal until some solution(s) is (are) reached. This traversal from goal to goal can be represented by an AND/OR graph called the **goal graph of the domain**, or, simply, the goal graph. Each node in a goal graph corresponds to a goal, where the unshaded nodes denote goals that are not solutions, and the shaded nodes denote solutions. A **connector** in this graph is from a set of goals $G = \{g_{i_1}, g_{i_2}, \ldots g_{i_n}\}$ to a goal g, where $g \notin G$. A circular mark in the connector near g indicates an AND edge, and that goal g can be inferred from goal g_{i_1} and goal g_{i_2} and ... goal g_{i_n}. Each node g_{i_j} is said to be a **parent** of node g. Moreover, a node g_1 is an **ancestor** of node g_2 iff g_1 is a parent of g_2, or g_1 is a parent of an ancestor of

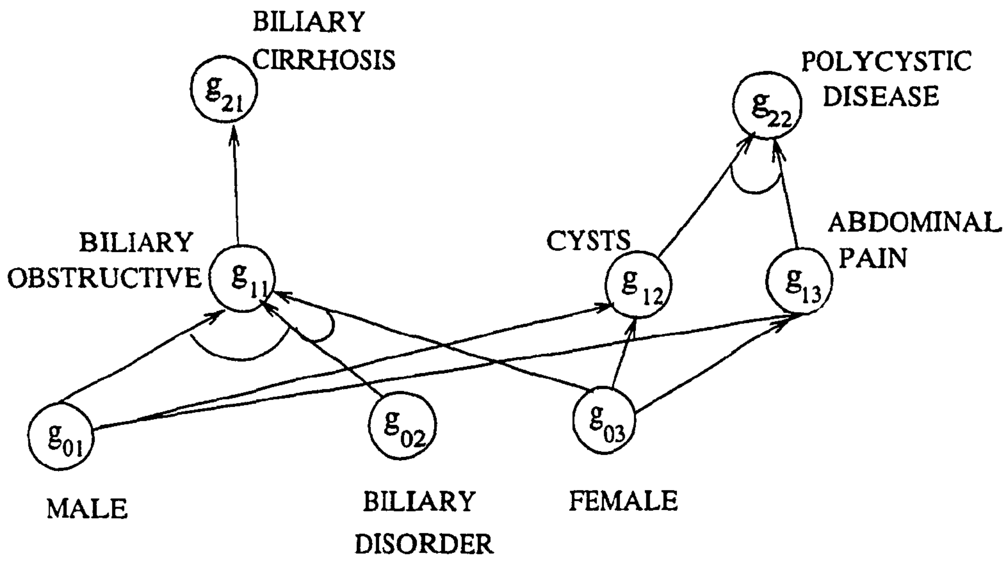


Figure 2.3: *A sample goal graph from a medical domain (to diagnose liver and biliary disorders) to abstract important states associated with problem solving.*

g_2. The different connectors to a goal portray the OR edges, and depict the different alternatives for inferring that goal. A goal graph that shows how goal inference would typically occur for the goals identified in Figure 2.1 is shown in Figure 2.3.

During knowledge acquisition, only the individual goal-to-goal requirements are recorded for rule encoding. In practice, a goal graph is extracted from a rule base to inspect the transitivity of the goal-to-goal progressions from initial evidence to a final goal (as implemented in the rule base). This can subsequently be used during evaluation to perform an empirical validation of the system (confirming the correctness of the extracted goal graph with a domain expert).

Every goal in a goal graph is associated with a **level**. Permissible combinations of initial evidence are said to be level-0 goals. For any other goal g, its level is defined to be one more than the level of its deepest parent, that is,

$$level(g) = 1 + \text{maximum of the level of all the parents of } g$$

The goals at level-1 are those goals that can be inferred starting from only initial evidence. Note, level-1 solution nodes indicate default solutions [Shinghal, 1992]. Solving any problem in the domain is equivalent inferring some level-1 goals followed by inferring deeper level goals, and so on until the solution is inferred. Thus, solving a problem is traversing the goal graph from level-0 goals to one or more final goal(s).

Goal-based problem solving specifies *what* is required to solve problems in a domain, but does not specify *how* to implement the specification. This is called the **functional view** of problem solving because it portrays how problems in the domain can be solved using goals, but hides the details of how goals are actually inferred in a system. This view of goal-based problem solving is not only suitable for development, but also reverse engineering. A systematic identification of goals to be achieved and the tasks to be solved can be used for restructuring an existing system, say to improve its performance. More specifically, the goal specification approach has been used successfully to restructure and re-design an existing rule-based system in a real application [Chander et al., 1995b]. The restructuring is described in the next section, and the re-design in chapter 3 (section 3.2).

2.3 Optimizing Rule Base Performance: A Case Study

The use of intelligent approaches to assist users in web search and browsing is becoming increasingly popular among multi-media researchers [Lieberman, 1995]. An application is currently being developed at this university to facilitate search and retrieval of documents in the world wide web [Chander et al., 1995b]. In this application, the system is given a set of input search fields associated with a document such as `title`, `author`, `subject`, ..., up to a total of thirteen fields; typically, a subset of these is entered by the user. The system then checks for the location of the document using a data base system, and having obtained the location, it eventually retrieves the document. The system contains a rule base that encodes knowledge of a reference librarian in order to provide appropriate help to a user of the application.

A group of reference librarians were consulted to acquire the domain knowledge. During implementation, the knowledge obtained from the the domain experts was translated directly to handle the various input fields while searching for a relevant document. One such rule is shown in Figure 2.4. This rule is coded to be activated

```
(defrule five-fields-data-not-match
 (declare (salience 91))
 ?addr1 <- (phase read ?type1 $?data1)
 ?addr2 <- (phase read ?type2&~?type1 $?data2)
 ?addr3 <- (phase read ?type3&~?type1&~?type2 $?data3)
 ?addr4 <- (phase read ?type4&~?type1&~?type2&~?type3 $?data4)
 ?addr5 <- (phase read ?type5&~?type1&~?type2&~?type3&~?type4 $?data5)
 (not (phase read ?type&~?type1&~?type2&~?type3&~?type4&~?type5$?));/*P*/
 (not (match-field $?))
=>
 (retract ?addr1 ?addr2 ?addr3 ?addr4 ?addr5)
 (assert (error-mess Data do not match.))
)
```

Figure 2.4: *An example rule from the rule base of the library expert system.*

when the user enters only five input fields to identify a document, but they do not match any of the existing document descriptions. The (first) prototype version of this system uses a sample library data base at a central site. It was developed under the CLIPS expert system shell [Giarratano and Riley, 1993], and the rule base contained 205 rules.

When a direct translation of knowledge is done as shown in Figure 2.4 to check whether each input field is different from the other using the "&" and "~" operators of CLIPS, it can result in an enormous computational overhead [Chander et al., 1995b]. To understand this aspect, consider Figure 2.5 that shows a typical (simplified) rule found in the rule base of the library expert. The rule tries to find a match when the user enters three input fields. But, this type of encoding increases enormously the pattern matching computation in the inference engine. For the rule shown in Figure 2.5, if the working memory contains {`(phase author)`, `(phase title)`, `(phase subject)`}, then the rule can fire not once, but six times. This can be ascertained from its trace shown in Figure 2.6. However, this is not the intent of the designer because the rule is supposed to fire once only. This behavior occurs because the encoding shown in Figure 2.5 caused the inference engine to find a matching instantiation for the variables `x`, `y`, and `z` in the rule antecedent for every possible

```
(defrule test-rule ;; A typical rule simplified from its original form
  (phase ?x)
  (phase ?y&~?x)
  (phase ?z&~?y&~?x)
=>
  (printout t ?x ?y ?z) ; some action
)
```

Figure 2.5: *A simplified version of a rule in the rule base of the library expert system.*

permutation of the given input.

When the type of rule encoding is of the form shown in Figure 2.5, rules that handle i input fields subsume the rules that handle $(i+1)$ input fields; however, these fields should be handled independently, that is, they are mutually exclusive. Rule subsumption arises because there are no atom(s) encoded in the rule antecedents to explicitly discriminate i input fields from j input fields, where $i < j$. Though subsumption between the rules can be avoided by explicitly checking if there are exactly i input entries for rule groups handling i input fields (see the pattern marked P in Figure 2.4), the pattern matching computation of the inference engine would still enumerate all possible permutation of the input fields (for every rule in the rule base) to decide which of the rules can fire. As a result, a huge amount of computation is spent on pattern matching to check if a rule's antecedent is satisfied, when the number of fields that are input is close to 13 (the total number of input fields). Though the facility of *retraction* provided by CLIPS—removing facts from working memory—can be used to prevent multiple firings of a rule, it however, cannot control the pattern matching computation, or the order in which rules are fired. Thus, when translating the rule in Figure 2.4 and others internally, the inference engine incurs an enormous amount of computation for pattern matching. The complexity of this computation increases combinatorially because the inference engine enumerates every possible permutation of the input fields for instantiating the variables in rule antecedents [Chander et al., 1995b, 1995c]. As a final note, the use of *salience* (a form of rule priority assignment in CLIPS) to force sequentiality so that $(i+1)$ input fields have priority over i input fields can interfere with the CLIPS rule execution mechanism

```
CLIPS> (matches testrule)
Matches for Pattern 1
f-1
f-2
f-3
Matches for Pattern 2
f-1
f-2
f-3
Matches for Pattern 3
f-1
f-2
f-3
Partial matches for CEs 1 - 2        +-------------+
f-3,f-2                              | Activations |
f-3,f-1                              | f-3,f-2,f-1 |
f-1,f-3                              | f-3,f-1,f-2 |
f-2,f-3                              | f-1,f-3,f-2 |
f-2,f-1                              | f-2,f-3,f-1 |
f-1,f-2                              | f-1,f-2,f-3 |
Partial matches for CEs 1 - 3        | f-2,f-1,f-3 |
f-3,f-2,f-1                          +-------------+
f-3,f-1,f-2
f-1,f-3,f-2
f-2,f-3,f-1
f-1,f-2,f-3
f-2,f-1,f-3
```

Figure 2.6: *Run time analysis of the sample rule shown in Figure 2.5. The facts in the working memory are* `f-1: (phase author)`, `f-2: (phase title)`, *and* `f-3: (phase subject)`. *The specific values associated with the fields are not shown for simplicity. The abbreviation* `CE n-m` *refers to a combination of facts that (partially) satisfy the antecedent. The activations refer to the number of times the rule can fire. Note that the number of activations of the rule is 6 (= 3!).*

making the run time behavior of the system harder to understand [Giarratano and Riley, 1993].

In the rule base of the library expert, rule groups similar to the one shown in Figure 2.4 were coded for each input combination. Rule subsumption was controlled inelegantly using additional patterns on rule antecedents (such as P in Figure 2.4); rule execution was controlled by rule saliences; and multiple firings of a rule were prevented using retraction. However, these "patches" had no control over the pattern matching computation. Rather, they made the rule base and the run time behavior of the system harder to understand [Chander et al., 1995b]. In fact, the system does not even *reset*—the CLIPS operation that determines the first set of rules that can fire after translating the rules internally—when the number of fields input is more than six.

Goal specification was applied successfully for optimizing the rule base of this system [Chander et al., 1995b]. An analysis of the knowledge encoding in rules such as the one in Figure 2.4 indicated incorrect and inadequate design of atoms to reflect the captured knowledge. We need to identify predicates and goals (lev 0 in this case) to reflect the extent of input handling. In addition, we also need to make use of the fact that rules that handle i input fields should be treated exclusively from rules handling a different number of input fields. Thus, we need two level-0 goals: `FIELD-INPUT(n)` that records how many fields were input, and `FIELDS(x,y,...)` that records the fields that were actually input. For example, if the fields `author`, `title` and `subject` are entered, then these atoms are respectively, `FIELD-INPUT(3)` and `FIELDS(author, title, subject)`, and must be input to the system. A restructuring of the example rule in Figure 2.5 is shown in Figure 2.7 (a), and its execution trace for the same input (augmented by the atom `FIELDS(author, title, subject)`) is shown in Figure 2.7 (b). With the added level-0 goals, the rule in Figure 2.4 can be restructured as shown in Figure 2.8.

The above modifications cut down the subsumption between rules due to the discriminating predicate `FIELD-INPUT`. The pattern matching computation was cut down because the inference engine need not enumerate every possible permutation of the input fields due to the predicate `FIELDS` that explicitly records which of the

```
(defrule testrule-2
(FIELDS ?x ?y ?z)
(phase ?x)
(phase ?y)
(phase ?z)
=>
(printout t ?x ?y ?z crlf)
)
```

(a) Restructured rule.

```
Partial matches for CEs 1 - 2
f-4,f-0
Partial matches for CEs 1 - 3
f-4,f-0,f-1
Partial matches for CEs 1 - 4
f-4,f-0,f-1,f-2
Activations
f-4,f-0,f-1,f-2
```

(b) Execution trace.

Figure 2.7: *Restructuring the rule shown in Figure 2.5.*

```
(defrule five-fields-data-not-match
   (field-input 5)
   (fields ?type1 ?type2 ?type3 ?type4 ?type5 $?)   ; $?  => ignore rest
   ?addr1 <- (phase read ?type1 $?data1)
   ?addr2 <- (phase read ?type2 $?data2)
   ?addr3 <- (phase read ?type3 $?data3)
   ?addr4 <- (phase read ?type4 $?data4)
   ?addr5 <- (phase read ?type5 $?data5)
   (not (match-field $?))
=>
   (retract ?addr1 ?addr2 ?addr3 ?addr4 ?addr5)
   (assert (error-mess Data do not match.))
)
```

Figure 2.8: *The restructured rule of Figure 2.4 based on additional level-0 goals.*

fields were actually input. This permitted a unique variable instantiation for rule antecedents, and the system was able to handle the full thirteen field input during its execution. In addition, the goal based restructuring also modularized the system better by explicitly grouping rules handling i input fields independent of rules handling j input fields, where $i \neq j$. This made the analysis of the rule base easier because each group typically contained no more than 7 to 8 rules. Such modularization of rule bases is important to improve their understandability and maintainability [Jacob and Froscher, 1990; Giarratano and Riley, 1993; Preece, Gokulchander, Grossner, and Radhakrishnan, 1993a]. The effect of goal-based restructuring on the rule base of the library expert system is shown in Figure 2.9.

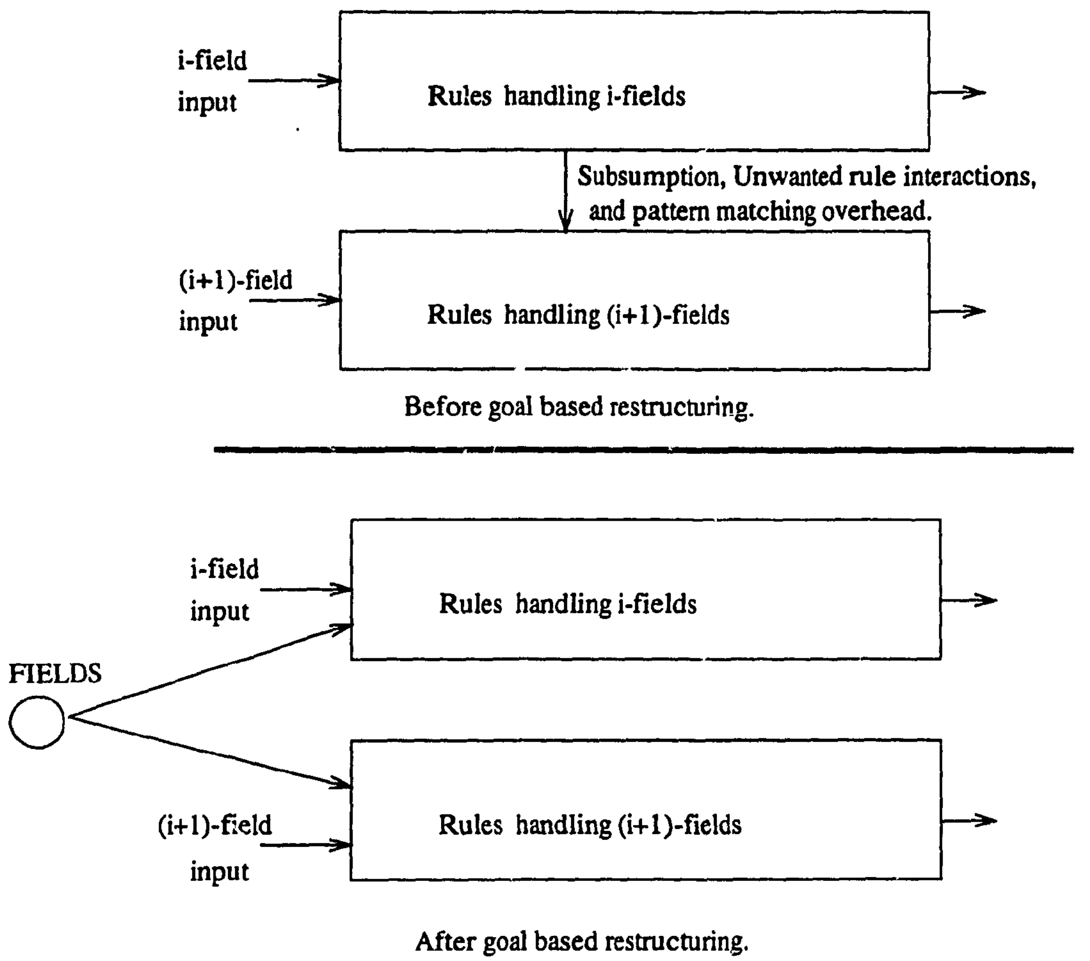

i-field input
Rules handling i-fields
Subsumption, Unwanted rule interactions, and pattern matching overhead.
(i+1)-field input
Rules handling (i+1)-fields
Before goal based restructuring.
i-field input
Rules handling i-fields
FIELDS
Rules handling (i+1)-fields
(i+1)-field input
After goal based restructuring.

Figure 2.9: *The effect of goal based restructuring on the library expert system. Individual, autonomous rule groups are formed consistent with the way problems would be handled in the domain.*

Chapter 3

Design Schemes for Rule-based Systems

The mapping between goals and their representation in a rule base is one-to-many and results in several *design schemes* for rule-based systems. The various design schemes impose different mapping restrictions for representing a goal in a rule base. Their use is described using a case study. In general, the applicability of a design scheme to a problem domain, and the relationships between the various design schemes are important factors that need to be considered before choosing a scheme for system development. In this thesis, a specific relationship between the design schemes called *inheritance* is explored. The relative pros and cons of using two schemes $\mathcal{D}_A$ and $\mathcal{D}_B$, having an inheritance relationship between them, for development versus maintenance are analyzed. This chapter also provides some metric measures to facilitate general comparison between various design schemes. Finally, an application of inheritance is outlined by considering the transformation of a rule base from one design scheme into another.

3.1 Rule-based System Design: Preliminaries

Conceptually, the design and development of a rule-based system can be viewed as a combination of comprehension, mapping, and encoding of the knowledge into a set of rules [Yost, 1993]. The comprehension is required in order to acquire the problem solving knowledge from the domain expert. The mapping is a formalization of this knowledge identifying the set of tasks associated with problem solving. The rule base development encodes the task domain representation of the acquired knowledge into a set of rules. Comprehension, mapping, and encoding are done by a knowledge engineer.

In this framework, knowledge is abstracted using goals (chapter 2). The next step is the mapping and representation of this knowledge based upon some restrictions that are satisfiable in the rule base to be developed. The design level formalizes this mapping.

In the implementation of a rule base, domain knowledge is encoded using a set of declarative rules of the following form [Shinghal, 1992]: $L_1 \wedge L_2 \wedge \ldots \rightarrow M_1 \wedge M_2 \wedge \ldots$, where the L's and M's are predicate logic atoms. The conjunction $L_1 \wedge L_2 \wedge \ldots$ is called the rule antecedent, and the conjunction $M_1 \wedge M_2 \wedge \ldots$ is called the rule consequent. Atoms in the consequent of a rule denote hypotheses. The syntax of the rules to encode the acquired knowledge partitions hypotheses inferred in a rule base into two types: intermediate and final. An intermediate hypothesis is one that occurs in the consequent of at least one rule and in the antecedent of at least one rule. A final hypothesis is one that occurs in the consequent of at least one rule, but never in the antecedent of a rule.

A design issue arises in mapping the intermediate and final goals arrived at during specification (chapter 2) to the intermediate and final hypotheses in a rule base. For example, while every final hypothesis is expected to be a goal atom (designed to form part of one or more solutions), the same need not be true for intermediate hypotheses. More generally, the design issue can be stated as follows:

Let $f = A1 \wedge A2 \wedge \ldots$ be a final goal and $i = A1' \wedge A2' \wedge \ldots$ be an intermediate goal specified, where the A's are hypotheses. The question *arises: what properties should $A1, A2, \ldots$ and $A1', A2', \ldots$ satisfy in the rule base?*

Since every goal atom can be encoded as an intermediate hypothesis, or a final hypothesis in a rule base, it is apparent that the mapping restriction imposed between goals and hypotheses is not unique. A precise description of this mapping restriction is necessary because it explicates the link between the semantic conception of the acquired knowledge (using goals) to its actual syntactic representation (using rule encoding). Such a description is called a **design scheme** for a rule base (or, simply, design scheme). More precisely, a design scheme is a satisfiable restriction imposed in mapping intermediate and final goals to intermediate and final hypotheses in the rule base.

Definition 2 (Design Scheme) *A design scheme $\mathcal{D}$ is an ordered pair of partial mappings $< \mu_1, \mu_2 >$ such that*

$$\begin{aligned} \mu_1 : F &\mapsto H_i \cup H_f \\ \mu_2 : I &\mapsto H_i \cup H_f \end{aligned}$$

where F is the set of final goals, I is the set of intermediate goals, H_i is the set of intermediate hypotheses, and H_f is the set of final hypotheses. The mapping is partial because not all hypotheses need be goal constituents.

The possible choices for the constituent hypotheses of final and intermediate goals are summarized in Figure 3.1. Below, the notation <Fn, Im> is used to denote a design scheme in analyzing the relationships between the the 25 possible design schemes from Figure 3.1.

While all the choices in Figure 3.1 are satisfiable restrictions resulting in various design schemes, some of them can be counter intuitive. For example, consider restriction I1 for intermediate goals that requires them to be composed of only final hypotheses. In this case, the intermediate goals cannot help infer any final goals because final hypotheses cannot be causal to other final hypotheses. A similar observation is applicable to restriction F4 for final goals. Thus, owing to counter intuitive

Hypotheses in a final goal f	Hypotheses in an intermediate goal i
(F1) All are final hypotheses.	(I1) All are final hypotheses.
(F2) At least one hypothesis is final.	(I2) At least one hypothesis is final.
(F3) At least one hypothesis is intermediate.	(I3) At least one hypothesis is intermediate.
(F4) All are intermediate hypotheses.	(I4) All are intermediate hypotheses.
(F5) No constraints.	(I5) No constraints.

Figure 3.1: *The choices for constituent hypotheses in a final goal f and an intermediate goal i, provided neither f nor i contains an inviolable.*

semantics of the specification, we recommend avoiding all design schemes with either of these mapping restrictions. Further, restriction I5, where intermediate goal composition is unconstrained, is discouraged because specification of intermediate goals can be uncontrolled, and such ad hoc specification may not reflect the intent of the domain expert. We, however, allow restriction F5 for ease in solution specification to facilitate functional, structural and/or empirical validation [Batarekh et al., 1991; Preece et al., 1994; Zlatareva and Preece, 1994], but care should be exercised in specifying solutions when using this restriction. Note, the constraints imposed by a design scheme has a bearing on the understandability and maintainability of a system (section 3.3.2).

A rule base built to encode the knowledge of goal-to-goal progression for a domain would contain rules that infer goal atoms. A set of rules ρ in the rule base is said to **realize** a given goal $g = A_1 \wedge A_2 \wedge \ldots \wedge A_n$, iff the consequent of these rules collectively contain the atoms $A_1, A_2, \ldots, A_n$. For example, consider constructing a rule base to identify a person's occupation in a university. One can identify final goals {`DEAN(x)`, `PROFESSOR(x)`}, and intermediate goals {`ADMINISTRATOR(x)`, `ACAD_OFFICER(x)`} associated with the domain. A set of rules that realize these goals is shown in Figure 3.2.

However, a set of rules realizing a goal does not necessarily imply that the goal would be inferred at run time. Thus, *goal realization* refers to a static aspect of knowledge manipulation (encoding) that takes place during rule base development, while *goal inference* refers to a dynamic aspect of knowledge manipulation (inference) that takes place at run time. For example, if the input to the rule base in Figure 3.2

R_1	`WORKS(x, Deansoffice)` $\rightarrow$ `ACAD-OFFICER(x)`
R_2	`ACAD-OFFICER(x)` $\wedge$ `STAFF(x, Deansoffice)` $\rightarrow$ `ADMINISTRATOR(x)`
R_3	`ACAD-OFFICER(x)` $\wedge$ `ELECTED(x, Board)` $\rightarrow$ `DEAN(x)`
R_4	`ADMINISTRATOR(x)` $\wedge$ `INCHARGE(x, Registration)` $\rightarrow$ `COORDINATOR(x)`
R_5	`WORKS(x, d)` $\wedge$ `FACULTY(x, d)` $\wedge$ `HASPHD(x)` $\rightarrow$ `PROF(x)`
R_6	`PROF(x)` $\wedge$ `TENURED(x)` $\wedge$ `HELDPOST(x, yr)` $\wedge$ `GT(yr, 8)` $\rightarrow$ `PROFESSOR(x)`

Figure 3.2: *Illustrating goal realization in a rule base.*

contains only {`WORKS(Henry, Deansoffice)`, `ELECTED(Henry, Board)`}, then the system would infer only one (final) goal: `DEAN(Henry)`.

When a set of rules ρ in a rule base $\mathcal{R}$ realizing a subset of the specified goals satisfies the goal mapping restrictions of some design scheme $\mathcal{D}$, then the rule base $\mathcal{R}$ is said to **partly adhere** to the design scheme $\mathcal{D}$. If a rule base as a whole satisfies the mapping restrictions imposed by a design scheme $\mathcal{D}$, then the rule base is said to **fully adhere** (or, simply, adhere) to that design scheme. For example, in Figure 3.2 the final goals `DEAN(x)` and `PROFESSOR(x)` are realized using only final hypotheses, and the intermediate goals `ACAD_OFFICER(x)` and `ADMINISTRATOR(x)` are realized using only intermediate hypotheses. Thus, the set of rules in Figure 3.2 satisfies the restrictions of design scheme <F1, I4>, and hence, it adheres to that scheme.

Definition 3 (Adhering to a design scheme) *For a given a goal specification $\mathcal{G}$, a rule base $\mathcal{R}$ is said to adhere to a design scheme $\mathcal{D}$, iff every intermediate (final) goal realized in $\mathcal{R}$ satisfies the intermediate (final) goal mapping restriction imposed by $\mathcal{D}$. A rule base $\mathcal{R}$ fully adhering to a design scheme $\mathcal{D}$ is denoted as follows:*

$$\mathcal{R} \models D$$

where, $X \models Y$ should be read as "In model X, properties of Y hold."

If only a subset ρ of a rule base $\mathcal{R}$ satisfies the restrictions of $\mathcal{D}$ (that is, $\mathcal{R}$ partly adheres to $\mathcal{D}$), it is denoted by $\rho \subset \mathcal{R} \models \mathcal{D}$. Note, it is possible for a rule base to simultaneously adhere to more than one design scheme: for example, the rule base in Figure 3.2 adheres to design schemes <F1, I3> and <F1, I4>. We elaborate more on this aspect in section 3.3.

The use of goal-based design to improve the performance of the library expert system is described in chapter 2 (section 2.3). However, as the rule base is to be embedded as part of a larger system, the overhead of the rule base due to its size, and the overhead of embedding the CLIPS run time control were found to be unacceptable. The rule base needed to be re-designed to optimize its size, performance, and the integration overhead. The re-design is described next.

3.2 The Library Expert: Re-design

The expert advice offered by a librarian should be offered to a user of this application both for searching and cataloging. In searching for a given set of documents, often the user offers vague, partial and/or incorrect information in his/her attempt to identify the terms used for the various descriptors of the index for the documents for which the user is searching. In other words, search specification by the user is often "ill-structured" [Simon, 1973]; hence, expertise is needed to help users to articulate their needs. In addition, the total number of input field combinations for document search to be handled is large. Only some of these input field combinations occur at any given time. For example, in a given search the user may be interested only in the documents of a particular subject hierarchy (`AI.ExpertSystems.Verification`) without caring for the title of the document. In this case, it does not make sense to consider search situations that also explicitly require the title of documents. Thus, by isolating and encoding the input combinations and handling them in rules, only a subset of the rules need be active to process the user input and provide appropriate help. This also improves the structure of the developed system in terms of its modularity and understandability [Jacob and Froscher, 1990; Preece et al., 1993a; Antoniou and Wachsmuth, 1994; Chander et al., 1995b].

Clearly, a direct encoding of knowledge to check for all input field combinations one after another would be quite inefficient (chapter 2, section 2.3). Further, in the embedded expert system, modeling the expertise of a reference librarian should not impose a major integration overhead.

In cataloging a new resource, the cataloging librarian uses the knowledge of the accepted norms for classification. From this knowledge, and his/her perception of the resource to be cataloged, the librarian chooses terms to describe the resource.

Reference librarians are aware of the conventions used by catalogers as well. They are typically familiar with the classification schemes, terms, index structures, and resources available in the domain of the user's need. The expertise of a reference librarian should be replicated to assist the users of our application in both searching and cataloging. The rules for searching are also applicable for cataloging because searching is required for cataloging as well.

3.2.1 System Requirements for Searching Virtual Libraries

The expert system for searching virtual libraries should be designed in a way that the query generated for document search should facilitate efficient database access, and reduce the number of incorrect results generated. This requires several aspects to be considered as part of its function. They are broadly classified below (more specific details are described in section 3.2.2):

1. **Context Sensitive Help** Context sensitive help should be provided while the user is entering the search fields: for example, the system should give appropriate aids to a user for completing a given field entry based on the current contents of the other related search fields.

2. **Query Result Analysis** The system should provide appropriate analyses when no documents are found by suggesting other alternatives to the user input, or when a large number of documents are found (the search query formulated was too general) by suggesting alternatives to focus the search.

3. **Cataloging and Information Organization** For cataloging, the expert system would be used for choosing appropriate terms from the thesaurus, and the associated keywords. Depending on the type of resource, the expert system should help the user scan the resource to try and fill-in other fields describing the document. Finally, the system should check that the resource being cataloged can be accessed by other users.

In addition, the expert system should not be a significant overhead in itself. Thus, we are constrained to keep the system small, but a useful and efficient aid to searching.

Though several expert system shells such as CLIPS facilitate knowledge engineering and rule encoding, they can incur a significant overhead when integrated into the overall system. There are several reasons for such an overhead:

1. The expert system shell has to be embedded in the applic~~~~tion to process and apply the rules because rule firings are controlled only by the inference engine. This can increase the size of the overall system considerably.

2. The inference engine overhead can be unacceptable at times. For example, for every rule firing, if a system such as CLIPS is used, its inference engine would recompute the set of rules that can fire. However, by knowing the contexts where a subset of rules can be applied, this inference engine overhead can be avoided.

3. The working memory of an expert system directly affects its performance. Typically, several hundred objects can qualify as targets by a search query. We need this intermediate hits, if the user needs to pursue the resources discovered by a subset of the query parameters. If this information accumulates in the working memory, the performance can slow down. This adds to the size of the rule base because we also need additional "clean up" rules to periodically cleanse the working memory from unwanted, or outdated information.

4. Inputs entered by a user and stored in data structures should be converted into appropriate formats for inserting into the working memory before the rules can fire. Later, rule inferences in the form of atoms should be parsed and mapped back into data structures. This increases the development effort and the overall complexity (overhead) of using an expert system.

5. The experimental system developed using CLIPS proved to be large (more than 100 rules), and the integration overhead was unacceptable (chapter 2, section 2.3).

6. It was also noted by our experience (section 2.3, chapter 1) that instead of lumping all the rules in a single rule base, if a rule application context is exercised using a rule (or, a set of rules), then performance improves significantly.

Once the rules were designed, we decided to encode them directly as C functions, and invoke a set of rules explicitly from specific parts of the system, thus, implicitly implementing the inference engine as part of the system. This also avoided the overhead of embedding an external shell into the system.

Using this approach, the "intelligence" of a cataloging/reference librarian, captured using a set of rules, was distributed into the system rather than being centralized in one place. This significantly improved the performance because only a small number of rules are applied at any time.

3.2.2 Implementation Details

The user interface for searching requires the user to input information that will help identify the documents distributed over the world wide web. The user entry can be at different levels of detail, and depending upon the level of detail entered, one of the tasks of the expert system is to provide the required amount of help to complete the input. This amounts to a reference librarian guiding a novice user in entering data based on what has already been entered by the user. For example, if the user has entered the subject `Computer Science`, then the help for other fields of the subject hierarchy would be tailored to `Computer Science`. Later, if the user changes the subject to, say, `Chemistry`, then help information would change accordingly.

The context sensitivity of help is complicated because the user can input a synonym for an input field. In general, synonyms can be entered for any of the fields corresponding to the subject hierarchy: a three level hierarchy for document classification. Moreover, a synonym entered at one level of the subject hierarchy may resolve in one or more levels. A synonym must be resolved using a control-thesaurus so that it can match with an appropriate document identification keyword. For example, the user can enter the synonym `KBS` which can mean `Knowledge Base Systems`, `Expert Systems`, or `Deductive Data Base Systems` that are part of the control-thesaurus. In general, searching the subject hierarchy for a control vocabulary can be modeled as a graph (tree) traversal. Synonyms, however, complicate this search for a control-thesaurus term because they make a tree traversal into a directed acyclic graph traversal, which is computationally more demanding. Yet, synonym resolution allows the system to automatically fill in higher levels of the subject hierarchy as they can be determined once a synonym at a lower level is resolved in the control-thesaurus. This greatly aids focusing the search to a specific set of documents. For example, if entry `KBS` for sub-subject was resolved to `Expert Systems`, then the subject of the entry would be automatically updated to contain `Artificial Intelligence` under which `Expert Systems` is cataloged. The synonyms and the associated control-thesaurus are kept in a local database.

Complication in synonym resolution arises when the user enters a synonym for subject that is actually a sub-subject synonym. Such synonyms are said to be *non-contextual.* On detecting a non-contextual synonym entered for a field, the system warns the user of this mismatch and resolves the synonym by traversing up (or, down) the subject hierarchy. The expert system then displays a list of control-thesaurus items found for the user to choose. For example, if the user enters a sub-subject synonym `RDB` that resolves to `Relational Data Bases` and `Remote Debugging` as sub-subjects (but not into any subject) in the entry for subject, the system warns the user and traverses up the subject hierarchy in this case to display the subjects corresponding to the sub-subjects `Relational Data Bases` and `Remote Debugging`.

When a synonym resolves at multiple levels of the subject hierarchy, the synonym is said to be *overloaded.* For example, nothing prevents a generic synonym such as `DB` to be used as a shorthand for the sub-subject and the sub-sub-subject levels of the subject hierarchy. Suppose as a sub-subject it resolves to {`Data Base Systems, Knowledge Base Systems`}, and as a sub-sub-subject it resolves to {`Relational Data Bases, Horn Clause Systems`}. Then, the context of the input entry is used to resolve that synonym for the level it corresponds to instead of resolving it into all matching terms. Handling non-contextual and overloaded synonyms mimic the discerning capability of a reference librarian in interpreting user input appropriately.

A complication arises when the user enters *partial values*: a sub-string for subject, for example `Data Bas`. Though one can display a list of subjects that have this substring, context sensitivity implies that the values already entered in other fields must also be considered in providing a help response to the user. Thus, the help would be based on not only the partial values of the current field, but also on existing values of other related fields. For example, if the user has entered `Hybrid relational` in sub-subject and `Frame` in sub-sub-subject field, then the context sensitive help for subject would take into account the current values in sub-subject and sub-sub-subject fields before providing appropriate help to the user. If the current values of sub-subject and sub-sub-subject entries are ignored, the user would be provided with a long list of subjects, many of which would have sub-subjects and sub-sub-subjects that may not match with the current values entered in these respective fields. The above action is equivalent to capturing a reference librarian's mental view to help focus the search for documents. The values in the other related fields need not be full values, but can themselves be partial values. Context sensitivity in the expert system gives only the appropriate amount of help that would be needed at a particular time.

When a partial value matches entries in multiple levels, then a *collision* is said to have occurred in the expansion of this partial value. In this situation, the system expands the substring appropriately on the level in which it was entered. If, however, no match can be found on the hierarchy based on the given input value, a warning is issued and the system displays a set of hierarchies to be chosen containing the given value at any level.

An interesting aspect of the expertise is *automatic inferencing capability.* A reference librarian would be able to identify that the user is searching under the subject `AI` if the user tells the librarian that he/she is searching for documents about `Expert systems`. In the system, once the value entered for a particular field is complete, but the other related fields are empty, the pertinent values of these fields are inferred and automatically updated with those appropriate entries. Such automatic inferencing of other related fields help to not only focus the search query to select a smaller number of documents, but also optimize the query.

The final aspect of the system is *warnings.* Whenever a user-entered field does not match the existing field values, the user is warned because this could result in a search query whose processing would not produce any document retrieval. Suppose

Expert Systems as a sub-subject entry does not match with Chemistry as a general subject, then the user must be warned of this potential mismatch. This does not mean that specific expert systems cannot exist in chemistry; all it means is that the topic of Expert Systems is not recorded as a sub-subject of chemistry. This mimics the cautionary advice that a reference librarian would give to a user.

We have developed a small demonstration version of the Expert System for rule testing, refinement, and for eventually evolving it into a complete system. The user interface of the expert system was coded under Motif; the rules themselves are implemented in C. The rules are grouped according to their category each associated with a set of goals to provide help to the user to formulate an efficient search query. The various rule categories and their associated goals (in parentheses) are as follows: context sensitive help category (synonym resolution, partial value expansion, warnings), search focus category (automatic inferencing), and search analysis category (error detection, warnings, field refinement). This classification does not mean the categories are exclusive: as a set of rules in one category are applied, they can later interact with rules in another category. For example, when a sub-string is expanded in a sub-sub-subject entry, if the sub-subject and/or subject entries can be filled, then they are automatically filled. This is due to the interaction between the rules and the goals they infer in the context-sensitive help category and the search-focus category. A simplified goal graph for the system is shown in Figure 3.3. The level-0 goals correspond to the various fields that are input by the user. For simplicity, the details of error checking, warnings, and non-context synonym resolution are omitted in Figure 3.3.

Presently, there are about 40 rules in the system (some additional rules may be needed during integration, but the total number of rules is not expected to exceed 50). Any given context requires the application of no more than 5 rules. A sample rule that handles the context sensitivity of a value entered for subject, if it is a synonym, is shown in Figure 3.4. The rule actually encodes two conceptual rules often applied by a reference librarian: one rule is meant for checking synonyms by the user for the subject field, and its resolution by displaying an appropriate list of subjects for the user to choose; the second one resolves a non-context synonym (such as a sub-subject

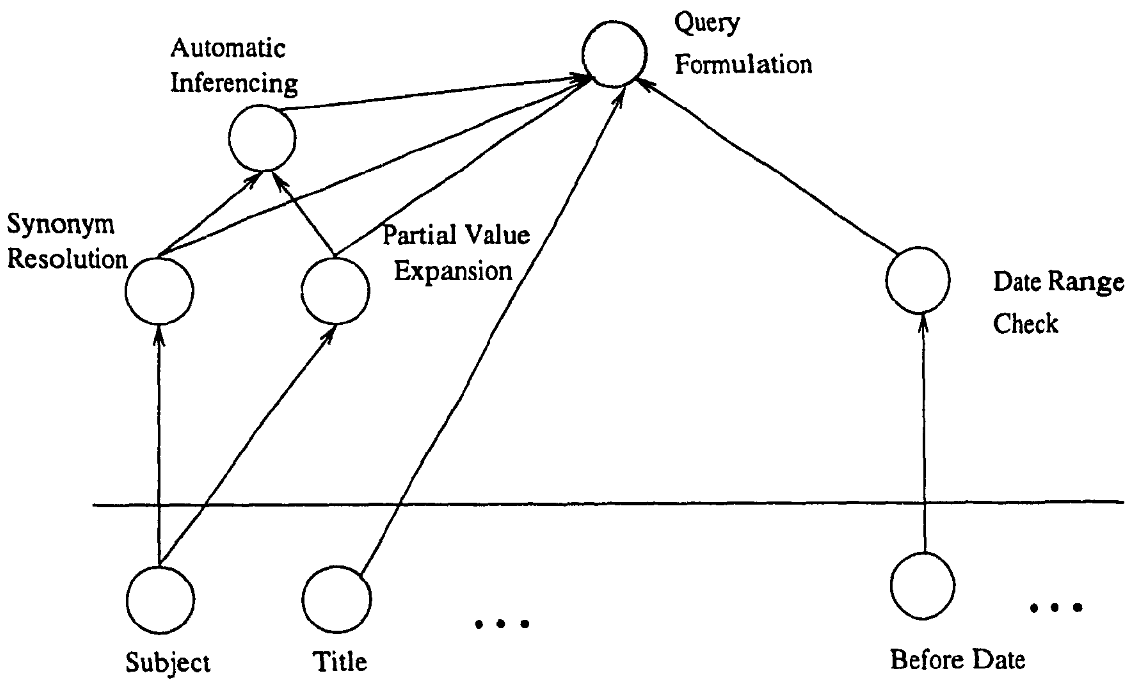


Figure 3.3: *A simplified goal graph depicting the various goals for the re-designed library expert system. For clarity, not all level-0 goals and connectors are shown.*

synonym entered in a subject field by mistake) and traverses the subject hierarchy to display a list of the corresponding subjects after issuing a warning.

The query formulation is based on the values of the fields entered by the user. The result of the query would be a set of documents, possibly empty, matching the search request of the user. Before formulating a search query, however, additional checking should be made: this is typical of the way a reference librarian would proceed to focus his/her search to identify a smaller number of documents. For example, ask the user to enter author information, title information, spelling correction, phonetic checks, and input checks such as mismatched subject hierarchy, etc[1]. If the result of the search query doc. not retrieve any document, then this is displayed as "No documents found" followed by an additional analysis of the inputs at the request of the user. Otherwise, the user is allowed to pick a subset of the retrieved documents for further manipulation (say viewing, or printing).

[1]The current version checks only for mismatched subject hierarchy, and incorrect date ranges.

```
/* context sensitive synonym resolution */
void Rule1(char *subject_string)
{
    synonym* syn;
    synonym *lev1, *lev2; /* level1 and level2 synonyms */

    /* rule 1: if synonym, then show list of corresponding subjects */
    /* rule 2: if a non-context synonym then show list of subjects by
        resolving the synonym and traversing the hierarchy as
        appropriate.
    */
    /* note:  synonym/subject hierarchy database access */
    syn = get_synonym(subject_string, 0); /* get ptr to approp
                                            synonym structure */
    if (syn)
      display_subjects(syn); /* display the subjects to pick one */
    else /* check if this is a level 1 or level2 synonym */
    {
        lev1 = get_synonym(subject_string, 1); /* note:  DB access */
        lev2 = get_synonym(subject_string, 2);
        if (lev1)
        {
            printf(" WARNING: Level-0 entry is a level-1 synonym\n");
            /* resolve lev1 and get and display the list of subjects
               obtained from the hierarchy.*/
            display_corresponding_subjects(lev1, 1); /* traverse */
        }
        if (lev2)
        {
            /* resolve lev2 and get and display the list of subjects
               obtained from the hierarchy. */
            printf(" WARNING: Level-0 entry is a level-2 synonym\n");
            display_corresponding_subjects(lev2, 2);  /* traverse */
        }
    }
}
```

Figure 3.4: *A sample rule in the expert system for context sensitive help associated with the general subject field in the user interface. Similar rules code other context sensitive help such as substring expansion, automatic inferencing, and warnings.*

3.3 Analyzing Properties of Design Schemes

There are theoretical and pragmatic aspects that should be considered in designing rule-based systems. A single design scheme may not be suitable for all domains; thus, we need a design framework. A design framework is characterized by the two following components: (1) a pragmatic component describing the qualitative aspects of the design schemes contained; and (2) a theoretical component that outlines the relationship between the design schemes to facilitate choosing a design scheme. More specifically, the pragmatic component dictates the choice between design schemes depending upon the relative importance given to development, and maintenance. The theoretical component allows for further compromises between development and maintenance by formalizing the properties of design schemes, and their use.

Unfortunately, there is no procedure to choose a design scheme from the set of possible design schemes (Figure 3.1, page 33) for a domain. In other words, for an arbitrary domain and development criteria, there is no known step-by-step method by which one can choose a design scheme from a set of design schemes. Often, the skill and experience of a knowledge engineer; size of the proposed system; the extent of analysis and synthesis components for problem solving used in the domain [Shinghal, 1992]; domain constraints expressed through domain dependent criteria; and long term objectives (such as expected level of performance and maintenance) expressed through domain independent criteria dictate whether to reject or select a design scheme from its characteristics [Debenham, 1992; Wells, 1993; Long and Neale, 1993]. A set of empirical criteria to guide the choice of a design scheme is discussed in section 3.3.3.

As an example, consider a domain dependent criteria where states accepted as solutions can also lead to (perhaps more refined) solutions: for example in a medical domain, if both `liver-disease` and `liver-cirrhosis` are acceptable solutions, note that the latter is a refinement of the former. For this domain, design schemes with choice F1 for final goals is not the right choice because solution causality cannot be captured by this choice for final goals. On the other hand, if one cannot infer additionally from domain solutions, or if the solutions are mutually exclusive then design schemes with choice F1 are perhaps more appropriate.

The above discussion is not meant to imply that the characteristics of the various design schemes are mutually exclusive. In Figure 3.1 some schemes subsume other schemes: for example, the <F1, I4> scheme is subsumed by the <F1, I3> scheme because every goal realized using the former scheme also satisfies the restrictions of the latter scheme. Though different degrees of subsumption can be identified between the design schemes, we consider only those instances when the intermediate and final goal restrictions in one scheme $\mathcal{D}_A$ are more specific than the corresponding restrictions in another scheme $\mathcal{D}_B$. The scheme $\mathcal{D}_B$ is then said to **inherit** the properties of $\mathcal{D}_A$. Inheritance simply means that the scheme $\mathcal{D}_B$ is more general (in terms of its restrictions) than $\mathcal{D}_A$; however, it does not imply that scheme $\mathcal{D}_B$ allows encoding of knowledge that cannot be encoded using $\mathcal{D}_A$.

Definition 4 (Scheme Inheritance) *A design scheme $\mathcal{D}_B$ is said to inherit from design scheme $\mathcal{D}_A$, iff every rule base $\mathcal{R}_A$ adhering to scheme $\mathcal{D}_A$ also adheres to scheme $\mathcal{D}_B$. Thus, if scheme $\mathcal{D}_B$ inherits from scheme $\mathcal{D}_A$, then the following is valid:*

$$(\forall \mathcal{R}_A)\ (\mathcal{R}_A \models \mathcal{D}_A) \ \Rightarrow\ (\mathcal{R}_A \models \mathcal{D}_B)$$

For simplicity, inheritance between two schemes $\mathcal{D}_A$ and $\mathcal{D}_B$, whenever the latter inherits from the former is denoted by $\mathcal{D}_B \models \mathcal{D}_A$. Note, a rule base adhering to scheme $\mathcal{D}_B$ need not adhere to scheme $\mathcal{D}_A$; thus, the converse of definition 4 need not be true.

Inheritance between schemes allows us to state that every property that holds in a rule base adhering to the <F1, I4> scheme also holds in at least one rule base adhering to the <F1, I3> scheme, but not conversely. This can be generalized between any two schemes that have an inheritance relationship between them.

Corollary 1 *If design schemes $\mathcal{D}_A$ and $\mathcal{D}_B$ are such that $\mathcal{D}_B$ inherits from $\mathcal{D}_A$, then every property that holds in a rule base $\mathcal{R}_A$ adhering to $\mathcal{D}_A$ can be made to hold in at least one rule base $\mathcal{R}_B$ that adheres to $\mathcal{D}_B$.*

Proof. *The proof trivially follows because $\mathcal{R}_A$ and $\mathcal{R}_B$ can be identical by virtue of scheme inheritance (definition 4).*

$$\begin{tikzcd}
\langle F1, I2\rangle \arrow[d] & \langle F3, I2\rangle \arrow[dd] & \langle F1, I4\rangle \arrow[d] \arrow[dr] & & \langle F3, I4\rangle \arrow[ddl] \arrow[dr] & \\
\langle F2, I2\rangle \arrow[dr] & & \langle F1, I3\rangle \arrow[d] & \langle F2, I4\rangle \arrow[dl] \arrow[d] & & \langle F3, I3\rangle \arrow[ddlll] \\
& \langle F5, I2\rangle & \langle F2, I3\rangle \arrow[d] & \langle F5, I4\rangle \arrow[dl] & & \\
& & \langle F5, I3\rangle & & &
\end{tikzcd}$$

Figure 3.5: *The various design schemes and their inheritance relationship. An arc from scheme $\mathcal{D}_A$ to scheme $\mathcal{D}_B$ indicates that scheme $\mathcal{D}_B$ inherits the properties of scheme $\mathcal{D}_A$.*

This corollary becomes important when analyzing the qualities of design schemes (section 3.3.2), and rule base transformation issues (section 3.4).

The different schemes provide varying amounts of freedom to encode knowledge. A particular scheme, though easier to implement, need not result in a rule base that is easier to maintain [Debenham, 1992] because the different schemes exhibit different qualities depending upon the flexibility allowed in realizing goals. Hence, the extent of understandability, the extent with which anomalies can creep into a system owing to incremental modifications, and the extent a scheme allows uncontrolled and ad hoc changes during rule modifications, differ in rule bases adhering to different schemes. In addition, the ability to choose a scheme that is flexible for development, but *transform* the rule base to adhere to another scheme that is favorable for maintenance can be useful to optimize both development and maintenance costs [Chander et al., 1995a]. All these aspects are affected by inheritance as will be apparent later. More specifically, inheritance affects the various qualities of the design schemes, and the ability to automatically transform a rule base adhering to one scheme into another. The inheritance between the various allowable design schemes is shown in Figure 3.5.

Below, we characterize how inheritance influences system maintenance by analyzing the following cost and quality issues:

- the relative costs involved in maintenance operations between design schemes $\mathcal{D}_A$ and $\mathcal{D}_B$, where $\mathcal{D}_B$ inherits from $\mathcal{D}_A$ (section 3.3.1); and
- quantify certain subjective qualities of the various schemes to compare those that have an inheritance relationship between them (section 3.3.2).

These analyses serve as guidelines to developers in choosing a scheme for a domain. We will then use the analyses as motivation to describe the rule base transformation process and its details (section 3.4).

3.3.1 Analyzing the Cost of Maintenance Operations

Using a set of general system parameters, it is possible to analyze the cost of the various commonly applicable maintenance operations. In particular, we compare the costs involved in maintenance operations using design schemes $\mathcal{D}_A$ and $\mathcal{D}_B$, where $\mathcal{D}_B$ inherits from $\mathcal{D}_A$. The typical operations that can arise during maintenance are summarized below:

(a) Adding a hypothesis to a goal;

(b) Deleting a hypothesis from a goal;

(c) Changing the hypothesis type in a goal;

(d) Modifying (increasing/decreasing) the arity of a hypothesis in a goal; and

(e) Changing goal types.

It is assumed that the operations above are listed in the order of decreasing frequency of occurrence. Thus, adding and deleting hypotheses to and from goals (respectively) occurs more frequently than changing goal types. For brevity, we will analyze the cost of only two of these maintenance operations in detail: adding hypotheses to a goal and deleting hypotheses from a goal. Not only these operations occur frequently, but the other maintenance operations can be analyzed in terms of these two operations [Chander, 1995]. The cost associated with the other operations is only summarized; the details can be found in [Chander, 1995].

For illustration, we choose design scheme $\mathcal{D}_A$ to be <F1, I4>, and design scheme $\mathcal{D}_B$ to be <F1, I3>, and discuss the costs and issues involved in the first two maintenance operations below. The above choice of design schemes was made because F1 is the most stringent restriction involving final goal realization, and I4 is the most stringent restriction involving intermediate goal realization. The various parameters used for the cost analysis are given below.

- C is the cost involved per unit of work (effort) expended in modification operations;
- $N_{h'}$ is the number of rules associated with hypothesis h' (that is, use h' and/or infer h');
- θ is the effort associated with checking and updating the design documentation;
- $|H_i|$ ($|H_f|$) is the number of intermediate (final) hypotheses;
- $|G_i|$ ($|G_f|$) is the number of intermediate (final) goals;
- N_M is the number of rules realizing M final goals (in the worst case, N_M = number of rules inferring final hypotheses); and
- $N_{M'}$ is the number of rules realizing M' intermediate goals.

Cost Analysis of Adding Hypotheses to a Goal There are two cases to consider because a hypothesis can be added to an intermediate goal, and/or a final goal:

1. Adding a hypothesis h to a final goal g: For both the <F1, I4>, and <F1, I3> schemes, owing to restriction F1, the hypothesis type must be checked to ensure that it is a final hypothesis from the documentation.
2. Adding a hypothesis h' to an intermediate goal g': In the <F1, I4> scheme, owing to restriction I4, only an intermediate hypothesis can be added. However, the flexibility of <F1, I3> scheme allows unconstrained modification to g' with no effort because if the realization of intermediate goal g' satisfies restriction I3,

Operation result	Cost in <F1, I4> scheme	Cost in <F1, I3> scheme
Cannot Add	$N_{h'} * C$	$N_{h'} * C$
Can Add	$N_{h'} * C$	$N_{h'} * C$
h' not in rule base	θ	θ

Figure 3.6: *Cost analysis for adding a hypothesis h' to an intermediate goal g'.*

then so is $g' \wedge h'$ for any hypothesis h'. However, this flexibility also implies uncontrolled and adhoc changes can take place on intermediate goal compositions without violating scheme restrictions.

The cost for operations for adding hypotheses in the <F1, I4> and <F1, I3> schemes are given in Figure 3.6. The first column in Figure 3.6 is the result of trying to add an hypothesis, and the other two columns give the associated cost in the two schemes. By similar analysis, the cost for adding a hypothesis h to a final goal can vary from θ to $N_h * C$. The cost and effort associated with the addition operation, thus, do not differ appreciably in the two schemes under consideration.

Cost Analysis of Deleting Hypotheses from a Goal A **unitary** goal is a goal with a single hypothesis. For example, $g' = h'$ is a unitary goal. If a goal has more than one hypotheses, then it is said to be a non-unitary goal. One of the issues in deletion is that a unitary goal becomes empty (when the sole hypothesis constituting the goal is deleted). Another issue that should be considered is that the deletion of a hypothesis from a goal can cause a scheme violation: the goal specification and its realization no longer adheres to the design scheme restrictions.

The cost of deleting a hypothesis h from a final goal is the same in the two example schemes under consideration and varies from θ to $C*N_h$. In deleting a hypothesis from final goals, no scheme violation checks need to be made. However, for intermediate goals, one must also check for scheme violation in the <F1, I3> scheme, but not in the <F1, I4> scheme because restriction I4 requires all intermediate goals to contain only intermediate hypotheses. The procedure is shown in Figure 3.7.

The cost analysis of the operations associated with deletion of an hypothesis h' from an intermediate goal g' is shown in Figure 3.8. In a typical rule base, the number of intermediate hypotheses is much larger than the number of final hypotheses. Thus, if C1 $= N_M * C$, C2 $= |G_f| * C + N_M * C$, and C3 $= |H_i| * C$, then $C3 > C2 > C1$. Note

```
<F1, I4>  scheme.
-----------------
Case 1. h' is a final hypothesis. This case is not applicable in this
scheme.

Case 2. h' is an intermediate hypothesis.
Effort requires checking the rules associated with h' only.

  2.1 If, however, g' is a unitary goal (that is, becomes empty
      because of deletion), then there are two choices:
     2.1.1) Add an intermediate hypothesis to g' from the set of
            intermediate hypotheses.
     2.1.2) Delete goal g' from the goal specification.

<F1, I3> scheme.
----------------
Case 1.     h' is a final hypothesis.
   Hypothesis h' can be deleted from g' after checking the rules
inferring h'.

Case 2.     h' is an intermediate hypothesis.

    2.1) If the hypothesis deletion  does not cause a scheme violation
         then the cost is the same as in <F1, I4> scheme.

    2.2) If g' is a unitary goal (that is, becomes empty after
    deletion) then, the same analysis as in <F1, I4> scheme above is
    applicable.

    2.3) If h' is the only intermediate hypothesis in (a non-unitary
         goal) g', then to prevent a scheme violation there are four
         choices.
      2.3.1) Add an intermediate hypothesis to g' from the set of
             intermediate hypotheses.
      2.3.2) Delete goal g' from the goal specification.
      2.3.3) Make one of the final hypothesis in g' to intermediate. This
             requires deciding which hypothesis to convert, and also
             revise some final goals (say M) containing this hypothesis.
      2.3.4) Make g' a final goal. This can require checking all the
             existing final goals (for subsumption), and changing
             rules realizing some of these goals in order to
             accommodate g' as a final goal.
```

Figure 3.7: *Deleting a hypothesis h' from an intermediate goal g'.*

Issue	Cost in <F1, I4> scheme	Cost in <F1, I3> scheme
Cannot delete		
	Not applicable (as operation cannot occur).	Not applicable, if scheme violation is not tolerated.
h' is final	Not applicable (as operation cannot occur).	$N_{h'} * C$
h' is intermediate	$N_{h'} * C$	$N_{h'} * C$, if no scheme violation occurs.
g' is unitary	delete goal (θ). Replace by another hypothesis ($\lvert H_i \rvert * C$).	delete goal (θ). Replace by another hypothesis ($\lvert H_i \rvert * C$).
Scheme violation	Not applicable.	Add another intermediate hypothesis ($\lvert H_i \rvert * C$). Delete g' (θ). Change g' to final goal ($\lvert G_f \rvert * C + N_M * C$). Make a hypothesis in g' intermediate ($N_M * C$).

Figure 3.8: *Cost analysis for deleting a hypothesis h' from an intermediate goal g'.*

that these costs would not occur in the <F1, I4> scheme because scheme violations do not occur as part of the deletion operation in this scheme. These additional costs arise because of the generality in the mapping restriction involving intermediate goals in the <F1, I3> scheme.

Summarizing Costs Involved in Other Maintenance Operations The operation of changing the type of a hypothesis in a goal g is the same as deleting a hypothesis from g and adding another. Thus, the above analysis for addition and deletion of hypotheses to and from goals can be combined. The operation of changing goal types should occur only when the domain knowledge has changed considerably, and can be treated as deleting all the hypotheses from a goal and adding a new goal (using the same set of hypotheses). The generality of the <F1, I3> scheme requires more checks to be made while performing this operation in general. Note, this operation can be quite complex, and can require deletion of hypotheses from *other* goals [Chander, 1995]. If the arity of a hypothesis in a goal is to be changed, then the example schemes incur the same cost provided the hypothesis is contained in a final goal: in this case, only rules realizing final goals have to be checked and modified, and the cost would be $N_M * C$ in the worst case. But, if the hypothesis is contained in an intermediate goal, then the <F1, I3> scheme incurs a higher cost because the entire rule base can require modification in the worst case. The details of these maintenance operations are described in [Chander, 1995].

In general, depending on the value of the cost parameters, the cost of a maintenance operation in schemes $\mathcal{D}_A$ and $\mathcal{D}_B$, where $\mathcal{D}_B$ inherits from $\mathcal{D}_A$ can be comparable in some situations (when scheme violations do not occur), but can differ widely in some situations (when scheme violations occur) with scheme $\mathcal{D}_B$ incurring additional costs for different choices in maintenance. This occurs owing to the generality of the goal-to-hypothesis mapping imposed by scheme $\mathcal{D}_B$, and the options that should be used in a maintenance operation. For the example schemes discussed above, the cost of adding hypotheses to a goal is comparable, whereas it can differ widely when deleting hypotheses from a goal: for instance, the deletion operation incurs additional costs in the <F1, I3> scheme, if scheme violations should be fixed by only rule modifications. Thus, a rule base transformation to scheme $\mathcal{D}_A$, where these costs can be less, is desirable for optimizing the maintenance costs [Chander et al., 1995a].

3.3.2 Assessing Qualities of Design Schemes

In this section, we develop some metric measures whose values can be used to compare certain subjective qualities (such as "understandability") between the different design schemes. The values of these metrics between schemes having an inheritance relationship is used as a further motivation for rule base transformation in the interest of quality improvement. The software engineering approach of using metric measures to model, and/or quantify rule-based system qualities has been a source of interest among researchers [Preece et al., 1993a; O'Neal and Edwards Jr., 1993; Chen and Suen, 1993; Mehrotra, 1995].

It is not uncommon in software engineering to quantify certain subjective qualities of a system based on its characteristics to facilitate comparison across different systems [Ghezzi et al., 1991; Conte, Dunsmore, and Shen, 1990]. For knowledge-based systems, it has been observed by researchers that the greater the deviation between the acquired knowledge and its representation, the lesser is the understandability of the system [Chandrasekaran, 1986; Yost and Newell, 1989]. In our case, a design scheme's restriction affects the underlying representation of the specified goals, and hence, that of the acquired knowledge. In addition, the extent a scheme can degrade owing to adhoc changes during maintenance is an indication of its susceptibility to become obscure and less maintainable.

A rule base is said to be clearly representing the structure of problems in the domain to the extent semantics of domain concepts used for problem solving are represented in the rule base using appropriate syntactic constructs [Chandrasekaran, 1986; Chander et al., 1995a]. In our case, the semantics of domain concepts are abstracted using intermediate and final goals. If an intermediate goal g is realized using only intermediate hypotheses in a rule base, then the rule base is said to better represent this goal than an intermediate goal g' realized using only final hypotheses. Clearly, the realization of goal g is more intuitive than the realization of g' because the semantic domain concept of an intermediate milepost is represented using the equivalent syntactic structure in the rule base.

Definition 5 (Non-corresponding construct in a goal) *A final (intermediate) hypothesis used in the realization of an intermediate (final) goal is said to be a non-corresponding construct in that goal.*

Thus, the extent a rule base represents intermediate and final goals using only the corresponding syntactic constructs (intermediate and final hypotheses, respectively) reflects how clearly the rule base represents the structure of the problems in the domain. We define these ideas using two distance metric measures: one for a goal and one for a rule base. The use of "distance-based" metric measures is not uncommon among researchers: for example in [Mehrotra, 1995], a distance metric is used to group related rules for improving the understandability of a rule base.

The goal distance metric can be used to measure the deviation of goal realization from its conceived semantics.

Definition 6 (Goal Distance Metric δ) *The distance between the conception and the* *realization of a specified goal g is defined as the number of non-corresponding constructs used for realizing the goal g in the rule base.*

Any value other than 0 for the distance metric of a goal represents that the goal is not realized using only the corresponding syntactic constructs in the rule base. As an example, let goal $g = h_1 \wedge h_2 \wedge h_3$ where h_1, h_2 are final hypotheses and h_3 is an intermediate hypothesis. If g is an intermediate goal, then $\delta(g) = 2$ because h_1 and h_2 are final hypotheses. If it is a final goal then $\delta(g) = 1$. Thus, the distance metric of intermediate (final) goals realized as a conjunction of only intermediate (final) hypotheses is 0.

The rule base distance metric measures the maximum deviation of a rule base from representing the domain concepts.

Definition 7 (Rule base distance metric Δ) *The distance metric of a rule base $\mathcal{R}$ is represented using the largest distance metric of the intermediate and final goals realized in the rule base. It is represented as an ordered pair $\Delta =< \delta_i, \delta_f >$ where,*

$$\begin{aligned} \delta_i &= MAX\ (\delta(g))\ (\forall g)\ \mathcal{R} \vdash g \quad \text{where, } g \text{ is an intermediate goal.} \\ \delta_f &= MAX\ (\delta(g))\ (\forall g)\ \mathcal{R} \vdash g \quad \text{where, } g \text{ is a final goal.} \end{aligned}$$

As an example, the largest distance metric for a rule base under scheme $< F1, I4 >$ would be $< 0, 0 >$, whereas a rule base under scheme $< F5, I3 >$ can have a distance metric as high as $< |H_f|, |H_i| >$, where $|H_i|$ and $|H_f|$ represent the number of intermediate and final hypotheses in the rule base respectively. These represent the maximum limits a rule base adhering to these respective schemes can deviate from representing an acquired concept using non-corresponding (hence, less intuitive) constructs; this arises as a result of the flexibility provided by a scheme. Figure 3.9 portrays the minimum and maximum value of the distance metric for a rule base adhering to the various design schemes. An alternative definition of the rule base distance metric would be to take the average of the distance metrics of the goals realized in the rule base.

A brief discussion on interpreting the values of these metrics appears below. A more detailed discussion and justification for such an interpretation can be found in [Chander, 1995].

Design scheme	Minimum	Maximum
$< F1, I4 >$	$< 0,0 >$	$< 0,0 >$
$< F1, I3 >$	$< 0,0 >$	$< \|H_f\|, 0 >$
$< F2, I3 >$	$< 0,0 >$	$< \|H_f\|, \|H_i\| >$
$< F5, I3 >$	$< 0,0 >$	$< \|H_f\|, \|H_i\| >$
$< F2, I4 >$	$< 0,0 >$	$< \|H_f\|, 0 >$
$< F5, I4 >$	$< 0,0 >$	$< \|H_f\|, 0 >$
$< F1, I2 >$	$< 0,1 >$	$< 0, \|H_i\| >$
$< F3, I2 >$	$< 1,1 >$	$< \|H_f\|, \|H_i\| >$
$< F2, I2 >$	$< 0,1 >$	$< \|H_f\|, \|H_i\| >$
$< F5, I2 >$	$< 0,1 >$	$< \|H_f\|, \|H_i\| >$
$< F3, I4 >$	$< 1,0 >$	$< \|H_f\|, 0 >$
$< F3, I3 >$	$< 1,0 >$	$< \|H_f\|, \|H_i\| >$

Figure 3.9: *The minimum and maximum values of the rule base distance metric for the various design schemes.*

1. *The distance metric of a goal can be interpreted as a measure of the relative difficulty in understanding the purpose and its realization of a goal in a rule base.*

   Justification. For example, final hypotheses in an intermediate goal g', though allowed in some schemes, cannot be causal towards any intermediate goal. In addition, confusion could arise regarding the purpose of realizing goal g', if none of the final hypotheses in g' are used in final goals. In general, if the distance metric of goals realized using a scheme is large, then the purpose and realization of these goals can be relatively more difficult to understand, unless the system is well documented.

2. *The range of the distance metric of a rule base, that is the values between its minimum and maximum values shown in Figure 3.9, is indicative of the amount of effort required to assess the impact of a rule base modification.*

   Justification. In particular, if the range is large enough, then a design scheme violation can occur in the rule base, yet it can go undetected. For example, the rule base distance metric for the <F1, I3> scheme that can vary from <0, 0> to <$|H_f|$, 0>. This can be interpreted as follows: assessing the impact of a modification, say changing a hypothesis in an intermediate goal into a final hypothesis, can require checking every rule inferring a final hypothesis in the worst case to ensure that this modification does not result in a scheme violation.

3. *From a maintainer's viewpoint, goal specification revisions over different versions of the system can be identified and made easily in a scheme that minimizes the goal distance metric.*

   Justification. This is based on the intuition that the more restrictive a scheme, the lesser will be the room for ambiguous (such as specifying the same goal as intermediate and final), or ad hoc specification of goals. This can eventually make the goal specification and the rule base obscure to understand and possibly error prone (see also the discussion in Figure 3.11, page 60).

4. *The larger the value of the distance metric of a goal, the more likely the goal contains redundant atoms.*

   Justification. For example, consider an intermediate goal $g' = h_1 \wedge h_2$ where h_2 is a final hypothesis. If hypothesis h_2 is not used in any final goal, then the question arises: does hypothesis h_2 and rules inferring h_2 indicate an error in the rule base and/or goal specification? A maintainer should therefore exercise care and additional effort when modifying rules that realize goal g'. Thus, a maintainer would prefer to choose a scheme that minimizes the distance metric of a goal to avoid these troublesome situations.

For convenience, the above discussion is summarized in Figure 3.10.

Inheritance between the schemes affects the values of these metrics as outlined by the following results.

Lemma 1 *If a design scheme $\mathcal{D}_B$ inherits from $\mathcal{D}_A$, then the maximum value of the distance metric of the a rule base $\mathcal{R}_B$ adhering to scheme $\mathcal{D}_B$ cannot be lower than that of a rule base $\mathcal{R}_A$ adhering to scheme $\mathcal{D}_A$.*

Proof. *Let the largest distance metrics of $\mathcal{R}_A$ be $\Delta_A =< \delta_i^A, \delta_f^A >$ and that of $\mathcal{R}_B$ be $\Delta_B =< \delta_i^B, \delta_f^B >$ respectively. We need to show that $\Delta_B \not< \Delta_A$. More specifically, we need to show that, $\delta_i^B \not< \delta_i^A$ and $\delta_f^B \not< \delta_f^A$.*

1. The distance metric of a goal can be interpreted as a measure of the difficulty in understanding the purpose and realization of a goal in a rule base.

2. The range of the distance metric of a rule base, that is the values between its minimum and maximum allowable values, is indicative of the amount of effort required to assess the impact of a rule base modification.

3. From a maintainer's viewpoint, goal revisions over different versions of the system can be identified and made easily in a scheme that minimizes the goal distance metric.

4. The larger the value of the distance metric of a goal, the more likely the goal contains redundant atoms.

Figure 3.10: *Interpreting the value of goal and rule base distance metric measures.*

Let g be the intermediate goal in scheme $\mathcal{D}_A$ such that $\delta(g) = \delta_i^A$. To have a value of δ_i^B smaller than δ_i^A, at least one of the intermediate hypotheses in g should be encoded as a final hypothesis in rule base $\mathcal{R}_A$ (without violating the restrictions of scheme $\mathcal{D}_A$) which should not be possible in $\mathcal{R}_B$. Let h be an intermediate hypothesis in g that is encoded in $\mathcal{R}_A$ as a final hypothesis, but cannot be encoded in $\mathcal{R}_B$ as a final hypothesis. However, if h is a final hypothesis in $\mathcal{R}_A$ and h cannot be encoded as a final hypothesis in any rule base $\mathcal{R}_B$ adhering to $\mathcal{D}_B$, then by corollary 1 (page 45) rule base $\mathcal{R}_A$ does not adhere to scheme $\mathcal{D}_B$. Thus, by definition of inheritance (page 45), it follows that $\mathcal{D}_B \not\models \mathcal{D}_A$. But, this contradicts our premise that $\mathcal{D}_B \models \mathcal{D}_A$. Thus, it is possible to encode h as a final hypothesis in $\mathcal{R}_B$ as well (without violating the restrictions of scheme $\mathcal{D}_B$). Hence, $\delta_i^B \geq \delta_i^A$. A similar (complementary) argument can be used to prove that $\delta_f^B \not< \delta_f^A$. See also Figure 3.9 for clarity.

The importance of the above lemma can be re-stated in English informally as "the overall understandability of a rule base adhering to a scheme $\mathcal{D}_A$ cannot be worse than a rule base adhering to scheme $\mathcal{D}_B$, whenever $\mathcal{D}_B$ inherits from $\mathcal{D}_A$." The above lemma holds even if the rule base distance metric is an average measure of the distance metrics of the goals realized.

Corollary 2 *The smallest value of the rule base distance metric of a scheme $\mathcal{D}$ is the smallest of the different schemes from which it can inherit.*

Proof. *This follows from Figure 3.9 and the inheritance graph (Figure 3.5).*

Corollary 2 can be restated in English informally as "the understandability of a scheme $\mathcal{D}$ even with the best possible encoding can always be achieved by at least one of the schemes from which $\mathcal{D}$ inherits."

It is well known from software engineering literature, the greater is the extent of ad hoc specification and development, the lesser is the maintainability of the system [Ghezzi et al., 1991]. This is true for rule-based systems as well because a rule base developed without designing proper constructs to represent knowledge is difficult to understand and maintain [Chander et al., 1995b; Yen et al., 1991; Jacob and Froscher, 1990; Mehrotra, 1995]. Thus, it would be useful if the extent of ad hoc modifications allowable in a scheme can be quantified. Consider the following metric definition that is based upon the flexibility of goal realization in a scheme.

Definition 8 (Adhocness of a goal g in a scheme $\mathcal{D}$) *The adhocness of a goal g in scheme D is defined as the ratio of the maximum number of non-corresponding constructs allowable in g to the least number of corresponding constructs allowable in goal g without violating the restrictions of $\mathcal{D}$. If $\delta_{max}(g)$ represents the maximum value of the goal distance metric of g, then*

$$Adhocness(g, \mathcal{D}) = \frac{\delta_{max}(g)}{m}$$

where m = minimum number of corresponding constructs allowable by $\mathcal{D}$ in g.

The above ratio can be interpreted as the extent of uncontrolled changes that can take place on the composition of a goal g and on the rules realizing that goal; uncontrolled changes can reduce the understandability and maintainability of a system. Note, it is possible to have rule bases adhering to schemes $\mathcal{D}_A$ and $\mathcal{D}_B$ respectively that have the same rule base distance metric, but the adhocness metric of the rule bases can vary: the variation represents the extent of uncontrolled changes that are allowed to happen during rule base modification in the respective schemes. A value of 0 for the adhocness metric indicates that a scheme forces a goal to be realized using only corresponding constructs, and the rigor of the scheme restrictions minimize the likelihood for adhoc changes to that goal. In contrast, a value of ∞ for the adhocness metric of a scheme indicates that the scheme restrictions do not impose any constraint on incremental rule base modifications; hence, errors can be easily introduced into a rule base by uncontrolled and hasty modifications. This can possibly deteriorate its performance and reliability (chapter 4, section 4.4). The adhocness metric associated with a scheme simply serves as a caution to a maintainer, when he/she modifies the system, to be careful lest the system degrades in its understandability, maintainability, and/or reliability.

As an illustration, the value of this metric for some of the design schemes is given below:

- <F1, I4> scheme: the final goal adhocness is 0.
- <F5, I3> scheme: the final goal adhocness is ∞.
- <F2, I3> scheme: the final goal adhocness is $|H_i|$ (the number of intermediate hypotheses in a rule base adhering to that scheme).

In order to distinguish between the distance metrics and the adhocness metric, an example is shown in Figure 3.11. The purpose of the adhocness metric associated with the various design schemes is to emphasize that a maintainer's natural inclination to reduce his/her modification effort in order to implement a change with ease (and minimal modifications to rules) can introduce errors and anomalies into the rule base. This scenario is more realistic in situations when the rule base is developed and maintained by different persons. The example shown in Figure 3.11 illustrates this point. In Figure 3.11, the flexibility of the chosen design scheme allowed the hypothesis $FLIES(x)$ in the final goal to be used up in an adhoc fashion. However, had the original scheme (<F1, I4>) been in effect, the hypothesis $FLIES(x)$ could not have been used in this adhoc fashion because restriction F1 constrains a given final goal to be realized using only final hypotheses. Thus, a greater rigor in goal realization imposed by a design scheme can reduce errors during modifications.

To summarize, the higher the value of the adhocness metric for a scheme, the more is the susceptibility of a rule base adhering to this scheme to uncontrolled changes permitted during modifications; this can eventually decrease its understandability,

The Adhocness metric and its relation to rule base modifications

Background: Consider a rule base to encode some bird characteristics. This encodes several attributes such as flying and non-flying birds, living characteristics, etc. A part of the rule base is shown in [a] below. This part of the rule base describes a typical characteristic of crows specifically. The rules realize a final goal $CROW(x) \wedge FLIES(x)$. Assume that the rule base R adheres to scheme <F1, I4> initially.

$\text{BIRD}(x) \rightarrow \text{BUILDNEST}(x) \wedge \text{FLIES}(x)$
$\text{BUILDNEST}(x) \rightarrow \text{LIVESONTREE}(x)$
$\text{LIVESONTREE}(x) \wedge \text{BLACK}(x) \rightarrow \text{CROW}(x)$
[a]

Evolution 1: Suppose a maintainer wishes to encode the knowledge in [a] more compactly, say to optimize performance. For this purpose, suppose another scheme $\mathcal{D}$ that is more flexible than the <F1, I4> scheme is to be chosen. Then, the adhocness metric of scheme $\mathcal{D}$ can be used as a measure in making such a decision because it indicates the extent of uncontrolled modifications possible during the incremental evolution of the rule base under that scheme. For illustration, suppose scheme <F5, I4> is chosen, then the adhocness metric of that scheme (∞ for final goals) indicates that totally uncontrolled modifications can take place on the rules realizing final goals. The incremental evolution of the rule base is shown in [b].

$\text{BIRD}(x) \rightarrow \text{BUILDNEST}(x) \wedge \text{FLIES}(x)$
$\text{FLIES}(x) \wedge \text{BLACK}(x) \rightarrow \text{LIVESONTREE}(x) \wedge \text{CROW}(x)$
[b]

Evolution 2: An additional set of changes to the rule base is necessitated when the knowledge fragments "some birds live on trees and fly" and "crows are black flying birds" were refined later to "all tree living birds fly" and "some flying birds are black crows." This required some re-writing of the rules in [b] and is shown in [c], while still adhering to the <F5, I4> scheme for rule encoding.

$\text{BIRD}(x) \rightarrow \text{LIVESONTREE}(x) \wedge \text{FLIES}(x)$
$\text{FLIES}(x) \rightarrow \text{BLACK}(x)$
$\text{FLIES}(x) \rightarrow \text{CROW}(x)$
[c]

However, the choice of scheme <F5, I4> for coding convenience has now resulted in anomalies in the rule base: the second rule in [c] is redundant, and the third rule is not universally true. Further, the information of nest building about birds is lost (missing knowledge in rule encoding is a common problem in rule base development [Preece, 1993]), while incorporating the refinement to the acquired knowledge, though all the three rule bases realize the given final goal $Crow(x) \wedge Flies(x)$.

Figure 3.11: *Illustrating the intent of the adhocness metric. Part [a] of the rule base in the figure initially adheres to <F1, I4> and <F5, I4> schemes. The parts [b] and [c] of the rule base show the extent of uncontrolled modifications that can take place during incremental development in the <F5, I4> scheme.*

maintainability, and reliability due to the introduction of errors and anomalies. In particular, if scheme $\mathcal{D}_B$ inherits from $\mathcal{D}_A$, then we can state the following:

$$(\forall g)\ \mathtt{Adhocness}(\mathcal{D}_B, g) \geq \mathtt{Adhocness}(\mathcal{D}_A, g)$$

The proof of this statement is similar to that of Lemma 1.

Thus, in the overall interest of reducing maintenance cost and general quality improvement, a scheme $\mathcal{D}_A$ from scheme $\mathcal{D}_B$ is preferable when $\mathcal{D}_B$ inherits from $\mathcal{D}_A$. However, $\mathcal{D}_A$ may not provide the ease in development as $\mathcal{D}_B$ does. Thus, one can adhere to scheme $\mathcal{D}_B$ during development, and later transform the rule base to adhere to $\mathcal{D}_A$ to facilitate maintenance operations, that is, optimize their cost. However, changing the rule base manually is cumbersome and the transformation is not always straight forward because the change can result in certain inconsistencies [Chander, 1995]. Thus, an automated way of transforming the rule base is considered desirable. The details of transforming a rule base adhering to one scheme into adhering to another scheme and the associated algorithms are discussed in section 3.4.

3.3.3 Empirical Criteria for Choosing a Design Scheme

Problem selection for expert system development requires consideration of several factors: some are domain dependent, while some are domain independent [Liebowitz, 1989]. Similarly, criteria to choose a design scheme from a set of schemes for a domain are not unique. In this section, we outline the various criteria that should be considered in general before choosing a design scheme for a domain. We do not claim that the criteria listed below are exhaustive. Rather, they reflect our learning from a retrospective view of our experiences in design and evaluation of rule-based systems [Preece et al., 1994; Chander et al., 1994; Chander, Radhakrishnan, and Shinghal, 1995; Chander et al., 1995b, 1995c, 1995a].

Criterion 1. The extent of analysis or synthesis aspects of problem solving [Shinghal, 1992; Chandrasekaran, 1986] associated with a domain is an important criteria. The extent scheme restrictions can accommodate analysis, synthesis, or, a mix of both type of problem solving varies. The schemes near the top in Figure 3.5 (page 46) favor analysis type of problem solving, whereas the schemes near the bottom can accommodate a mix of analysis and synthesis type of problem solving.

Criterion 2. The mental view held by a domain expert from whom knowledge is acquired plays a major role in scheme selection because this often dictates the representations to be chosen by a knowledge engineer for capturing this knowledge accurately [Chandrasekaran, 1986, 1983]. For example in a medical domain, a doctor infers a set of tests from a given set of symptoms. This is analysis type of problem solving because a new set of hypotheses are now generated from existing hypotheses. However, from the basis of one or more of the tests, if an intermediate disease is diagnosed, then this inference is always used in future conclusions, unless it is disproved; thus, there is also a synthetic type of problem solving until a final (set of) disease(s) are diagnosed. The extent a doctor uses analysis and synthesis in disease diagnosis should be reflected by the acquired knowledge. This in turn influences the choice of a scheme that is better suited to encode the acquired knowledge. Further, during maintenance, a different domain expert may be consulted before making some enhancements. Hence, the ease with which the newly acquired knowledge can be encoded into a system also affects the choice of a scheme. Note, even in a pure analysis type of domain, a domain experts mental view could involve some synthesis type of problem solving.

Criterion 3. The extent a scheme offers flexibility in goal realization is important because it can optimize development and evaluation costs. The cost of evaluation, during development and maintenance, cannot be overlooked. System evaluation issues, however, are non-trivial, and this often plays a deciding role in selecting a scheme (section 4.2, chapter 4). The schemes near the bottom in Figure 3.5 provide relatively more flexibility in development and system evaluation owing to their less stringent restrictions for goal realization.

Criterion 4. The extent a design scheme can improve the maintainability of a system varies. Hence, if long term usage of the system is important, then maintainability is an issue that cannot be overlooked. Design scheme qualities quantified using metrics in section 3.3.2, thus becomes an important factor because they are indicative of the understandability, maintainability, and the reliability of a system based on a design scheme.

Criterion 5. The solution causality in a domain is another consideration: if states that are accepted as solutions can also lead to (perhaps more refined) solutions, then design schemes with choice F1 for final goals is not the right choice for this domain. On the other hand, if solutions are mutually exclusive then design schemes with choice F1 are perhaps more appropriate. A similar argument can be made for intermediate goals: if intermediate states should also reflect partial solutions, then choice I2 of Figure 3.1 is more appropriate; thus, in domains such as network configuration management, choice I2 for intermediate goals is preferable. In domains where intermediate states cannot form partial solutions, but can only lead to solutions, choice I3 or I4 is preferable.

Criterion 6. The variability of knowledge in a domain impacts the choice of a design scheme. Variability of knowledge can be accommodated, with relative ease, by the design schemes near the bottom of Figure 3.5 owing to their less stringent restrictions in goal realization. Note, drastic changes to goals specified can be cumbersome to incorporate in a system using the schemes at the top in Figure 3.5.

As an example, consider constructing a rule base to identify a person's occupation in a university environment. The domain description appears in Figure 3.12, and a typical analysis for a knowledge engineer to prune, or choose design schemes is described below.

Applicability of the Design Schemes: The design schemes based on choice I1 or I5 for intermediate goals are ruled out according to the recommendation in section 3.1. In addition, for the above description choice I2 for intermediate goals that forces every intermediate goals to have at least one final hypothesis is not convenient for encoding. The <F1, I3>, and <F1, I4> schemes can accommodate the goal specification in Figure 3.12, but can have difficulty in accommodating goals specified later as the rule base develops incrementally. For example, adding new goals REGULAR_STUDENT, IRREGULAR_STUDENT, but emphasizing their relation to GRAD, UGRAD, while still retaining them as solutions, can be cumbersome owing to the restriction F1 that requires all final goals should contain only final hypothesis. Note, changing a final goal to an intermediate goal can require significant rule base modification in these schemes. The

Final Goals: GRAD, UGRAD, DEAN, ASSOCDEAN.
Intermediate Goals: FACULTY, STUDENT, POTLSTUDENT, ACAD_OFFICER, ADMINISTRATOR, ACADEMIC, ENROLLED.
Every admitted person is a potential student, and the university community broadly classifies persons into academics, academic officers, administrators,...based upon the initial evidence such as whether a person is admitted, registered, or works in an administrative office. The specificity of a particular classification leads to the identification of the person's occupation as under graduate, graduate, professor, associate dean,.... Some of the domain constraints are as follows: (1) a person can become a dean only after serving as an associate dean for at least 8 years; and (2) a person can hold more than one occupation on a part time basis.

Let us further suppose that two more final goals were specified at a later stage for the university domain: REGULAR_STUDENT and IRREGULAR_STUDENT to further refine the acquired knowledge. All under graduate and graduate students are considered regular, and the diploma students as irregular.

Figure 3.12: *An example description of the occupation of the various persons in a university domain.*

<F2, I3> scheme can accommodate the goal specification with ease. But, of course, care should be taken while maintaining causal/temporal relations between final goals. For instance, consider the domain constraint involving DEAN and ASSOCDEAN: we simply cannot realize final goal ASSOCDEAN as an intermediate hypothesis and use it to infer DEAN owing to restriction F2. A similar argument applies to the <F2, I4> scheme. However, a reversal of goal types is better accommodated in the <F2, I3> scheme owing to its flexibility. Realizing the solutions as they are specified while maintaining their relationships is easier in schemes with choice F3 for final goals that require every solution to have at least one intermediate hypothesis. However, some of the final goals in this scheme can be counter intuitive, if they are realized as only intermediate hypotheses (hence, less understandable). A similar observation applies to schemes with choice F5 for final goals that impose no constraints in final goal realization. In addition, care should be exercised if such a scheme is chosen because the rule base can become obscure and error prone due to incremental modifications (section 3.3.2). A knowledge engineer may thus prefer to choose either the <F2, I3> scheme, or the <F2, I4> scheme for this domain.

3.4 Rule Base Transformation Procedures

The transformation of a rule base adhering to a design scheme $\mathcal{D}_B$ into one that adheres to a scheme $\mathcal{D}_A$, where $\mathcal{D}_B$ inherits from $\mathcal{D}_A$ is not always straightforward because the converse of corollary 1 (page 45) need not always hold: that is, there can be rule bases that adhere to scheme $\mathcal{D}_B$, but not to scheme $\mathcal{D}_A$. Such a transformation, however, would be useful in some situations, if not all, from our discussions in sections 3.3.1 and 3.3.2. One way to transform the rule base is to check the rules and goals manually and make the required modifications. However, manual modification on the rule base is cumbersome, can introduce anomalies, and can change the existing dependency between the rules that can result in an incorrect operation. Thus, procedures to automate this transformation is desirable. These procedures would detect any scheme violations (with respect to $\mathcal{D}_A$) in a rule base adhering to $\mathcal{D}_B$ and perform rule re-writing to annul these violations, and revise goal specification if necessary. But, if the overheads of the transformation procedures outweigh their benefits, then such a transformation is not justified.

Although a rule base transformation is not meant for optimizing the performance of a rule base, it should not result in an unacceptable increase in the size of the rule base, or decreased performance. Performance would degrade considerably if a transformation results in an increased time for pattern matching (as the bulk of the time required for processing a rule base is spent on pattern matching). A transformation procedure, however, would not increase the pattern matching computation, if it introduces exactly one extra atom A_e for every atom A deleted in a rule antecedent and A_e contains exactly the same arguments as the original atom A. Since the pattern matching computation to check for the satisfaction of a rule antecedent is proportional to the number of atoms in the antecedent, the time for pattern matching will be unaffected. The transformation, however, can result in the generation of extra rules and hypotheses [Chander, 1995; Chander et al., 1995a].

In general, certain properties of a rule base must be preserved by any transformation procedure. These are called **transformation invariances**, or simply, invariances. They are outlined below:

1. The existing dependency between the rules should be preserved by a transformation; otherwise, incorrect operation, and thereby incorrect solution would result.

2. The set of atoms in a rule base accounts for the knowledge content in the rule base. Thus, every atom present in the rule base should be preserved along with its arguments. In particular, if the transformation of a rule r,

$$r : A(x,y) \wedge B(y) \rightarrow C(x,y)$$

   results in a rule r' such that,

$$r' : A(x) \wedge B(y) \rightarrow C(x,y)$$

   then the content of the rule base is not preserved.

3. A rule base transformation is an automated process. Thus, it should not interfere with the semantics of problem solving conceived and abstracted using intermediate and final goals.

4. The observed run time behavior should not change as a result of a transformation. In particular, the transformed rule base should infer the same solution(s) to a given input. Note, preserving the content of a rule base does not necessarily ensure that its run time behavior can be preserved. For example, if the transformation of a rule r,

$$r : A(x) \longrightarrow B(x)$$

   results in a rule r' such that

$$r' : B(x) \longrightarrow A(x)$$

   then, the transformation preserved the rule base content, but not its run time behavior. Finally, an example of a transformation that preserves neither a rule base content, nor its inference is shown in Figure 3.13

The invariances that should be maintained by a transformation (Θ) when transforming a rule base $\mathcal{R}_B$ (adhering to scheme $\mathcal{D}_B$) into a rule base $\mathcal{R}_A$ (adhering to scheme $\mathcal{D}_A$) are shown in Figure 3.14.

`Original rule:` $A(x) \wedge B(y) \rightarrow C(x,y) \wedge D(x)$
`Transformed rule:` $A(x) \wedge B(y) \rightarrow C(x,y)$

Figure 3 13: *A transformation that fails to preserve a rule base content and inference.*

The general principles behind a rule base transformation are listed below:

1. Annul scheme violations using rule re-writing and extra rules and hypotheses.
2. Preserve the dependency between the rules as it was existing before. This would preserve invariances 1, 2, and 4 of Figure 3.14.
3. Revise goal specification (if necessary) to preserve invariance 3 of Figure 3.14.

For illustration, let us consider transforming a rule base adhering to the design scheme <F1, I3> into one that adheres to <F1, I4>. In this case, the transformation requires re-writing of only those rules realizing intermediate goals because the final goal restriction is the same in both the cases. However, as a hypothesis in the rule base can only be intermediate or final, but not both, the transformation can encounter a conflicting situation called **discrepancy**.

Definition 9 (Discrepancy in a transformation) *When transforming a rule base adhering to scheme $\mathcal{D}_B$ into one that adheres to scheme $\mathcal{D}_A$ by converting hypothesis types in a goal in order to annul a design restriction in the target scheme $\mathcal{D}_A$, the transformation can inadvertently violate another restriction of that scheme. When this occurs, the transformation is said to have encountered a discrepancy.*

As an example, let g' be an intermediate goal and g be a final goal. Let, $g' = h' \wedge k_1$, and $g = k_1 \wedge k_2$, where h' is an intermediate hypothesis, and k_1 and k_2 are final hypotheses. Consider a rule base adhering to the <F1, I3> scheme (final goals have at least one final hypothesis and intermediate goals have at least one intermediate hypothesis). Though k_1 is a final hypothesis, h' is an intermediate hypothesis, and hence, intermediate goal g' satisfies restriction I3. However, in trying to transform rules realizing intermediate goal g' into scheme <F1, I4>, we have

1. **Structure Preservation:** The dependency between rules is preserved. More specifically, if $r_1, r_2 \in \mathcal{R}_B$ are such that r_1 is causal to r_2, then the corresponding rules in $\mathcal{R}_A$ should have the same causality relation. In other words,

$$(\forall r_1, r_2 \in \mathcal{R}_B)\ \mathcal{C}(r_1, r_2) \Rightarrow \mathcal{C}(\Theta(r_1), \Theta(r_2))$$

The causality relationship between two rules r_1 and r_2 in a rule base is denoted by $\mathcal{C}(r_1, r_2)$ above to indicate that the firing of r_1 can be causal towards the firing of r_2.

2. **Content Preservation:** Every hypothesis present in $\mathcal{R}_B$ is also present in $\mathcal{R}_A$. Thus, if a denotes a hypothesis

$$(\forall a)\ a \in \mathcal{R}_B \Rightarrow a \in \mathcal{R}_A$$

It follows that $\Theta(a) = a$ for every atom a. Thus, the knowledge represented in an atom should be preserved by a transformation.

3. **Specification Preservation:** Preserve specification and problem solving semantics as existed before. Every (revised) goal g_A for scheme $\mathcal{D}_A$ realized in $\mathcal{R}_A$ also satisfies the restrictions of scheme $\mathcal{D}_B$ without any type change. Thus for example, every intermediate goal realized in $\mathcal{R}_B$ is realized (after revision, if necessary) as intermediate goals in $\mathcal{R}_A$ without violating any scheme restrictions in $\mathcal{D}_A$, or $\mathcal{D}_B$. If the set of intermediate (final) goals for rule base $\mathcal{R}_B$ is denoted by I_B (F_B) then (using similar notations for $\mathcal{R}_A$),

$$\begin{aligned}(\forall g)\ g \in I_B &\Rightarrow \Theta(g) \in I_A, I_B \\ (\forall g)\ g \in F_B &\Rightarrow \Theta(g) \in F_A, F_B\end{aligned}$$

4. **Run Time Inference Preservation:** For an initial evidence input (say X), if $\mathcal{R}_B$ infers a set of hypotheses H, then the set of hypotheses inferred in $\mathcal{R}_A$ when X is input would contain H (the other hypotheses would be extra hypotheses, if any, generated by the transformation procedure.) Hence, the solutions inferred in $\mathcal{R}_A$ would be the same as that in $\mathcal{R}_B$ for a given input evidence X.

$$(\forall a, r \in \mathcal{R}_B)\ r \vdash a \Rightarrow (\exists \sigma \in \mathcal{R}_A)\ \sigma \vdash a \quad \text{and} \quad Fires(r) \Rightarrow Fires(\sigma)$$

where σ refers to a rule sequence, and the predicate $Fires(x)$ is true if rule or rule sequence x can fire. This invariance simply states that every atom inferred by a rule in $\mathcal{R}_B$ would also be inferred in $\mathcal{R}_A$, perhaps by a rule sequence (the sequence can contain a single rule). Note, content preservation, does not imply run time inference preservation, or vice versa.

Figure 3.14: *Invariances associated with a transformation when transforming a rule base $\mathcal{R}_B$ adhering to scheme $\mathcal{D}_B$ into a rule base $\mathcal{R}_A$ adhering to another scheme $\mathcal{D}_A$.*

```
/* 1: Option deletion for handling discrepancy */
Procedure  Transform_F1I3_to_F1I4-1;
begin
/* Only intermediate goals containing a final hypothesis are in
   violation */
    For every intermediate goal g'
        For every final hypothesis h in g'
              /* Goal revision if discrepancy occurs */
               IF h is contained in a final goal,
                   Delete h from g'.
                   IF goal g' becomes empty, remove g'
               ELSE Add a rule, /* annul violation using .. */
                  r': h -> h'    /* .. extra rule & hypothesis */
           /* Preserve dependency: no changes required */
/* For efficiency combine all extra rules generated into one rule */
end      {Transform_F1I3_to_F1I4-1}
```

Figure 3.15: *Using deletion while handling discrepancy.*

a problem: since <F1, I4> requires all intermediate goals to contain intermediate hypotheses, we can change the hypothesis type of k_1 to intermediate so that g' satisfies restriction I4. But, now final goal g that contains hypothesis k_1 violates restriction F1. The transformation is said to have reached a discrepancy: we cannot transform rules realizing intermediate goal g' without violating one of the scheme restrictions of <F1, I4> scheme. Hypotheses such as k_1 in the above example are called **offending** hypotheses.

There are three ways to handle a discrepancy:

Option 1 Delete the offending hypotheses, (see Figure 3.15). This is easiest to implement, but it can make the system less understandable and maintainable. This option is discouraged in general. Note, this option is not guaranteed to preserve the invariances outlined in Figure 3.14.

Option 2 This option minimizes changes to final goals, while revising some intermediate goals (if necessary) to fix a discrepancy (see Figure 3.16). This option preserves solutions as they were specified as much as possible.

```
/* 2: Preference to final goals; minimize changes to solutions as much
   as possible while revising goals */
Procedure Transform_F1I3_to_F1I4-2;
begin
/* Only intermediate goals containing a final hypothesis are in violation */
    For every intermediate goal g'
         For every final hypothesis h in g'
                /* Intermediate goal revision if goal discrepancy occurs */
                   Rewrite every rule that infers h of the form
                         r: Antec -> h AND Rest
                   into
                         r: Antec -> h' AND Rest /* extra hypothesis */
                  /* annul violation and Preserve dependency */
                     Add a rule
                         r': h' -> h  /* extra rule & hypothesis */
                         /* h can now be inferred from Antec as before */
                         Revise goal g' replacing h by h'
/* For efficiency combine all extra rules generated into one rule */
end      {Transform_F1I3_to_F1I4-2}
```

Figure 3.16: *Minimizing changes to final goals while handling discrepancy.*

Option 3 This option minimizes changes to intermediate goals (see Figure 3.17). This is a converse of option 2. From a view point of functional and/or empirical validation, we favor minimal changes to existing final goals. This option, however, can revise solutions with extra hypotheses that can result in a lack of understandability of the specified solutions and their intent.

Note, however, that options 2 and 3 retain the total number of goals constant. As an example, the rule base of Figure 3.18 is transformed into Figure 3.19 under option 1.

Theorem 1 *For every design scheme $\mathcal{D}_B$ that inherits from another scheme $\mathcal{D}_A$, there is a transformation procedure that can transform a rule base adhering to $\mathcal{D}_B$ into another that adheres to $\mathcal{D}_A$ preserving the invariances listed in Figure 3.14.*

Proof. *Let the rule base adhering scheme $\mathcal{D}_A$ be denoted by $\mathcal{R}_A$ and that of scheme $\mathcal{D}_B$ be denoted by $\mathcal{R}_B$. Let Θ be the transformation procedure that transforms rule base $\mathcal{R}_B$ into $\mathcal{R}_A$. Then, the proof consists in asserting that the following formula is*

```
/* 3: Preference to intermediate goals; minimize changes to
   intermediate goals as much as possible  while revising goals */
Procedure  Transform_F1I3_to_F1I4-3;
begin
/* Only intermediate goals containing a final hypothesis are in violation */
    For every intermediate goal g'
        For every final hypothesis h in g'
              /* Intermediate goal revision if goal discrepancy occurs */
               For every rule group of the form
                     r: Antec -> h AND Rest
               Add a rule,
                     r': h -> h'   /* extra rule & hypothesis */
              /* The goal g' now satisfies restriction I4. But, this
                 may cause some final goals to violate F1. Hence,
                 the following revision  may be needed */
               For every final goal g containing h,
                    Replace h by h'
            /* Preserve dependency: no changes required */
/* For efficiency combine all extra rules generated into one rule */
end      {Transform_F1I3_to_F1I4-3}
```

Figure 3.17: *Minimizing changes to intermediate goals while handling discrepancy.*

<F1, I3> scheme

Goal specification.

Intermediate goals	Final goals
$g'_1 = h'_1 \wedge h'_2 \wedge k_1$	$g_1 = k_1 \wedge k_2$
$g'_2 = h'_3 \wedge k_1 \wedge k_4 \wedge k_5$	$g_2 = k_1 \wedge k_3$

The rule base. The atoms a, b, c, d, e, f are initial evidence.

Rule #	Rule
1	$a \rightarrow h'_1 \wedge l'_1$
2	$l'_1 \rightarrow h'_2 \wedge l'_2$
3	$l'_1 \wedge b \rightarrow k_1$
4	$l'_2 \wedge c \rightarrow k_2$
5	$l'_1 \wedge l'_2 \rightarrow k_3$
6	$d \rightarrow h'_3$
7	$e \wedge h'_1 \wedge h'_2 \rightarrow h'_4$
8	$h'_4 \rightarrow k_4 \wedge h'_5$
9	$f \wedge h'_5 \rightarrow k_5$

Figure 3.18: *An example rule base and goal specification in <F1, I3>* scheme.

<F1, I4> scheme (transformed)

Discrepancy handling: Delete offending hypotheses.

(Revised) Goal specification.

Intermediate goals	Final goals
$g'_1 = h'_1 \wedge h'_2$	$g_1 = k_1 \wedge k_2$
$g'_2 = h'_3 \wedge k_4 \wedge k_5$	$g_2 = k_1 \wedge k_3$

The rule base. The atoms a, b, c, d, e, f are initial evidence.

Rule #	Rule
1	$a \rightarrow h'_1 \wedge l'_1$
2	$l'_1 \rightarrow h'_2 \wedge l'_2$
3	$l'_1 \wedge b \rightarrow k_1$
4	$l'_2 \wedge c \rightarrow k_2$
5	$l'_1 \wedge l'_2 \rightarrow k_3$
6	$d \rightarrow h'_3$
7	$e \wedge h'_1 \wedge h'_2 \rightarrow h'_4$
8	$h'_4 \rightarrow k_4 \wedge h'_5$
9	$f \wedge h'_5 \rightarrow k_5$
10	$k_4 \rightarrow x1$
11	$k_5 \rightarrow x2$

Figure 3.19: *The rule base of Figure 3.18 after transformation and revised goal specification. Rules 10 and 11 and hypotheses $x1$ and $x2$ represent extra rules and hypotheses generated by the transformation.*

valid.

$$\mathcal{D}_B \models \mathcal{D}_A \Leftrightarrow (\exists\Theta)\,(\Theta(\mathcal{R}_B) = \mathcal{R}_A)$$

To prove the validity of the above formula, we need to prove that rules in $\mathcal{R}_B$ that are in violation of the restriction(s) imposed by scheme $\mathcal{D}_A$ can be modified so that they no longer violate those restriction(s).

A rule can violate a design scheme restriction in only three ways:

Case 1. Having a goal atom h_f in its antecedent, or inferring a goal atom h_f that is in violation of a final goal restriction.

Case 2. Inferring a goal atom h_i or using atom h_i in its antecedent in violation of an intermediate goal restriction.

Case 3. A rule in violation of both final and intermediate goal restriction due to inferring and/or using atoms in its consequent and antecedent respectively.

For all the above cases, the rule base $\mathcal{R}_B$ can be made to satisfy the restrictions of scheme $\mathcal{D}_A$ by the following rule re-writing:

- For annuling design scheme violation(s) that can be caused by case 1 by using h_f in a rule antecedent, replace every occurrence of atom h_f by $d_f \wedge h_f$ in rule consequents, and replace every occurrence of h_f by d_f in rule antecedents. This would annul the violation with respect to restrictions F1, and F2 which are the only restrictions involving final hypothesis. If the violation occurs because atom h_f is a final hypothesis, then the violation is annulled by adding a rule $r_d : h_f \rightarrow d_f$.
- For annulling design scheme violation(s) due to case 2, the proof arguments are similar (to the one above), but complementary (as the violation involves intermediate goals).
- For annulling design scheme violation(s) that can be caused by case 3 which can cause a discrepancy, use rule rewriting in either (a) or (b) above to annul violations of final or intermediate goal restrictions respectively. Then perform goal revision to minimize changes to final goals (option 2, page 70), or intermediate goals (option 3, page 71), if necessary.

This preserves invariances 1, 2, and 4 of Figure 3.14 in general, but goal revisions (when handling a discrepancy, if any) can cause a goal to satisfy the restrictions of $\mathcal{D}_A$, or $\mathcal{D}_B$, but not both. However, this cannot happen because inheritance between $\mathcal{D}_A$ and $\mathcal{D}_B$ implies that every goal realized in $\mathcal{R}_A$ satisfying restrictions of scheme $\mathcal{D}_A$ would also satisfy the restrictions of scheme $\mathcal{D}_B$.

Lemma 2 *If a design scheme $\mathcal{D}_B$ does not inherit from $\mathcal{D}_A$, then a transformation of a rule base adhering to $\mathcal{D}_B$ to one adhering to $\mathcal{D}_A$ is not guaranteed to preserve all the invariances shown in Figure 3.14.*

Proof. *Assume to the contrary that there exists a transformation between two schemes that do not have an inheritance relationship, but preserves all the invariances in Figure 3.14. For example, consider design schemes <F1, I2> and <F3, I3>. There is no inheritance relationship between these schemes because final and intermediate goals allowable in scheme <F3, I3> is not necessarily allowable in scheme <F1, I2>. Of course, syntactic rule re-writing can preserve invariances 1, 2 and 4 of Figure 3.14. However, let us consider preserving invariance 3 of Figure 3.14. For example, consider a final goal $g = h$ in <F1, I2>. During transformation if h is made into an intermediate hypothesis (say using rule $h \rightarrow d$) to satisfy the restrictions of scheme <F3, I3>, it no longer satisfies the source scheme restrictions. However, g cannot be revised into $g = d$, as it would violate restriction F3. There is no way that goal g can satisfy the restrictions of both the schemes by rule re-writing. This violates invariance 3 associated with a transformation shown in Figure 3.14.*

If no discrepancy arises during a transformation, then the time for processing the rules can be kept the same as before even if extra rules are generated. In this case, the extra rules are generated in order to annul a design scheme violation. As they do not change the existing dependency between the rules, pragmatically speaking, the extra rules can be generated in a separate file, which need not be loaded at run time.[2] However, this need not be true when an extra rule is generated while fixing a discrepancy. The number of such extra rules generated is proportional to the number of goal atoms common to intermediate and final goals. This number should be minimized in a systematically formulated goal specification as explicated by the following corrollary.

Corollary 3 *No discrepancy would arise during a transformation if the set of atoms used in intermediate and final goals are disjoint.*

[2]Alternatively, they can be merged into one rule as indicated by the comments in the figures depicting the transformation procedures.

Proof. *A discrepancy arises only if a re-writing of rules realizing intermediate (final) goals violates a design restriction involving final (intermediate) goals as per the definition given in page 67. This can happen only for rules that infer atoms that are common to both intermediate and final goals. (See also the example following the definition of discrepancy in page 67 for clarity.)*

The hypotheses in the extra rules, other than the extra hypotheses, generated in a transformation indicate that these hypotheses were encoded using the "leniency" of scheme $\mathcal{D}_B$, and should be restructured (manually, if necessary) in the rule base to adhere to scheme $\mathcal{D}_A$ for reduced maintenance costs. Thus, the transformation procedures only partially mechanize the ease of maintenance operations; even such partial mechanization for maintenance can be quite important as observed in Debenham (1992).

As a final note, each of the thirteen directed edges in Figure 3.5 can be thought of as akin to a transformation procedure because each edge depicts an inheritance relationship between two design schemes. We described in detail only the transformation procedure to transform a rule base adhering to scheme <F1, I3> to one that adheres to scheme <F1, I4>. In Figure 3.20, we provide a brief and informal English description of the other transformation procedures for the sake of completeness. In the interest of space, actual procedural level details of preserving rule dependencies, atom replacements, extra rule generation are not described, but rather the essence of the transformation from one scheme into another. It is assumed that discrepancy, if arises, is handled by one of the options discussed in page 69. The procedural level details of the transformation can be found in [Chander, 1995].

From	To	Transformation Requirement
<F5, I2>	<F3, I2>	Convert one hypothesis in every final goal containing only final hypotheses into intermediate hypothesis.
<F5, I2>	<F2, I2>	Convert one hypothesis in every final goal containing only intermediate hypotheses into final hypothesis.
<F2, I2>	<F1, I2>	Similar to the one described in section 3.4, but no discrepancy can arise in this case.
<F5, I3>	<F5, I4>	Convert final hypothesis in intermediate goals into intermediate hypothesis.
<F5, I3>	<F3, I3>	Convert a final hypothesis in every final goal containing only final hypothesis into intermediate hypothesis.
<F3, I3>	<F3, I4>	Convert every final hypothesis in an intermediate goal into intermediate hypothesis; no discrepancy can arise.
<F5, I3>	<F2, I3>	Convert an intermediate hypothesis in a final goal containing only intermediate hypotheses into final hypothesis; discrepancy can arise unlike transformation between <F5, I2> and <F2, I2>.
<F5, I4>	<F3, I4>	Convert a final hypothesis in a final goal containing only final hypotheses into intermediate hypothesis.
<F5, I4>	<F2, I4>	Convert an intermediate hypothesis in every final goal containing only intermediate hypothesis into final hypothesis.
<F2, I3>	<F2, I4>	Similar to the one between <F3, I3> and <F3, I4>, but discrepancy can arise.
<F2, I3>	<F1, I3>	Convert every intermediate hypothesis in a final goal into final hypothesis.
<F2, I3>	<F2, I4>	Similar to the one between <F2, I3> and <F1, I3>.
<F1, I3>	<F1, I4>	Described in detail in section 3.4.

Figure 3.20: *Summarizing the effect of the other scheme transformations.*

Chapter 4

Rule Base Model and its Application to System Evaluation

The relationship between the rules in a rule base, causing them to interact for inferring goals, is described. Goals are inferred in a rule base using non-linear rule sequences called paths. Paths and goals can be used as basis for several evaluation procedures for rule-based systems. More specifically, validation, performance and quality assessment, and verification using paths and goals for rule-based systems are outlined. The associated tools and algorithms developed to facilitate evaluation are also described using a case study.

4.1 Modeling Goal Inference in a Rule Base

A rule base $\mathcal{R}$ contains a set of declarativ_ ules that encode the acquired knowledge of goal inference in a domain for problem solving [Shinghal, 1992; Giarratano and Riley, 1993], while adhering to the constraints set forth by the chosen design scheme. The execution of a set of rules and its associated control in a rule-based system $\mathcal{S}$, discussed in chapter 1 (section 1.1), can be represented as a triplet $< \mathcal{R}, \mathcal{I}, \mathcal{W} >$, where

- $\mathcal{R}$ is the set of rules in its rule base,
- $\mathcal{I}$ is the inference engine, and
- $\mathcal{W}$ is the working memory.

In the proposed model of a rule base, we are interested in capturing the rule interactions that occur while goals are being inferred. This is called the **structural view** of problem solving because this model explicitly describes *how* goals are inferred in a system for problem solving. This should be compared with the functional view of problem solving discussed in chapter 2 (section 2.2) that explicates *what* is required for problem solving. More specifically, the functional view of problem solving is traversing a goal graph using the connectors (section 2.2), whereas the structural view of problem solving is concerned with the representation of connectors using rules to infer goals in a rule base.

A rule becomes **enabled** to fire whenever its antecedent evaluates to true. If the rule does indeed fire, then the hypotheses in its consequent are said to be inferred. For a rule r to become enabled, the atoms in its antecedent other than initial evidence must have been inferred as hypotheses. Thus, some other rule(s) inferring those hypotheses must have fired. This general dependency between rules is captured by the notion of **rule attainability**.

Definition 10 (Attainability of a rule) *A rule r_2 is attainable from another rule r_1 iff an atom in the antecedent of r_2 is unifiable with an atom in the consequent of r_1, or with an atom in the consequent of a rule r that is attainable from r_1. The attainability between the rules r_1 and r_2 is denoted by $r_1 > r_2$.*

Though the rule attainability relation imposes a precedence between the rules, not all rules are comparable using this relationship. The reason is as follows: if rules r_2 and r_2' are attainable from r_1, then it does not necessarily imply that r_2' is attainable from r_2 (or, vice versa) because the atoms in the antecedent and consequent of these rules (respectively) can be disjoint. The rule attainability is thus said to impose a *non-linear* precedence relationship between the rules in a rule base.

The rule attainability relation is transitive by virtue of its definition, and the transitivity of this relation can be used to capture rule interactions that occur in general. Consider for example, a set of rules ρ in a rule base that are seemingly not related to one another because their antecedents and consequents are disjoint. By computing the transitive closure of the attainability relation for this rule base, the rule(s) attainable from the rules in ρ can be identified. This rule dependency explicates the purpose of (firing) the rules in ρ, and the causal effect of the interaction between these rules towards enabling some other rule(s) in the rule base. Thus, the transitive closure of the rule attainability relation is a measure of the extent to which rules interact in a rule base.

Using the attainability of a rule and level-0 goals that are composed of permissible combinations of initial evidence, the rule dependencies in a rule base can be modeled using a graph called a **rule graph**. A rule graph is a labeled directed graph. Each node in a rule graph corresponds to a rule. The atoms inferred by firing a rule r_i are shown by labeling the directed arcs from r_i as $A_i^1, A_i^2, \ldots$. An arc labeled A_i^n from r_i to r_j indicates that the atom A_i^n in the consequent of r_i unifies with an atom in the antecedent of r_j.

In a goal-based view of problem solving, we want to capture how rule dependencies result in inferring hypotheses whose conjunction constitutes a specified goal. The rule dependencies are thus not carried across goal atoms because inferring a goal atom can be used to infer a goal subsequently; goal atoms in the antecedent of a rule would be supplied by already inferred goals. We therefore consider only the dependency between rules that arises due to the unification of non-goal atoms. This is called the **accessibility** of a rule.

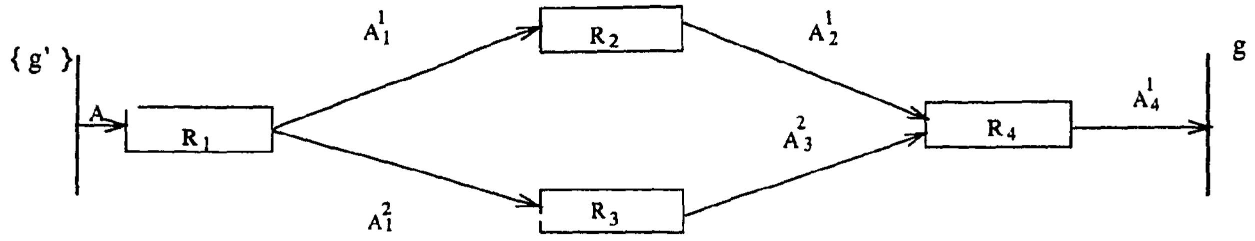
Definition 11 (Accessibility of a rule) *A rule r_2 is accessible from another rule r_1 iff a non-goal atom in the antecedent of r_2 is unifiable with an atom in the consequent of r_1, or with an atom in the consequent of a rule r that is accessible from r_1. The set of all rules from which a rule r is accessible is called the accessibility set of r. It is denoted by $\alpha(r)$.*

Clearly, the rule accessibility relation is non-linear, transitive, and is a subset of the rule attainability relation. The transitive closure of the accessibility relation is a measure of the interactions that occurs between rules *required to infer goals.*

The rule accessibility relation can be used to identify several sub-graphs of a rule graph such that the conjunction of the inferred goal atoms in these sub-graphs constitutes a specified goal. Thus, each of these sub-graphs depicts a connector in terms of a non-linear sequence of rules. More specifically, a connector in a goal graph, from a set of goals $G = \{g_{i_1}, g_{i_2}, \ldots g_{i_n}\}$ to goal g, where $g \notin G$, indicates a non-linear sequence of rules that must be fired to infer g from G. This sequence of rules is called a **rule base path** or, simply, path. Figure 4.1 illustrates a rule graph that is a path. The rule graph is enclosed between goals G and g (depicted using vertical bars) indicating the rules to be fired to infer g from G. An arc labeled A_i^n from r_i to r_j indicates that non-goal atom A_i^n in the consequent of r_i unifies with an atom in the antecedent of r_j (that is, rule r_j is accessible from r_i). The edge is labeled by the atom that unifies r_i and r_j: for example, the edge labeled A_1^2 portrays that rule R_3 is accessible from rule R_1 via that atom. If a rule infers a goal atom belonging to g, this is depicted as an edge that is incident on the vertical bar representing g (see edge labeled A_4^1). The conjunction of goal atoms inferred by the rules in a path must constitute a goal as specified for the problem domain. Goal graphs and rule graphs representing paths are compared in Figure 4.2.

Formally, every path is a set of rules with a precedence relationship defined between the rules. It is denoted by $< \Phi, \succ >$, where Φ is the set of rules in the path, and $\succ$ is a precedence relation between the rules of the path defined as follows:

$$(\forall\, r_i, r_j \in \Phi)\; r_i \succ r_j \Rightarrow r_j \text{ is accessible from } r_i$$



where the goal g = A_4^1 and G = { g' }

Figure 4.1: *A rule graph depicting a path.*

Goal Graph	Rule Graph
The sequence in which goals are inferred.	The sequence in which rules are fired.
An AND-OR graph.	A labeled directed graph.
Nodes represent goals, some of which are solutions.	Rectangular nodes represent rules. Vertical bars represent goals.
Connectors represent paths that portray order of inferring the goals.	Directed edges represent atoms inferred in rule firings.

Figure 4.2: *A comparison of the rule and goal graphs.*

A path can also be represented by specifying the precedence relationship $\succ$ between the rules in the path. Thus, the path of Figure 4.1 can be represented as $\succ = \{< R_1, R_2 >, < R_1, R_3 >, < R_2, R_4 >, < R_3, R_4 >\}$.

Rules in a path, however, must be enabled so that they can fire once the goals required by the path are inferred. For a rule r to be enabled, a subset of the accessibility set of r must have fired. While there can be many such subsets, we are interested in only some of them. The minimal set of rules that are required to enable a rule is called the **enabling set** of a rule [Grossner, Preece, Gokulchander, Radhakrishnan, and Suen, 1993].

Definition 12 (Enabling set of a rule) *For a rule r with accessibility set $\alpha(r)$, suppose there exist sets $\eta_1(r), \eta_2(r), \ldots$ such that*

(i) $\eta_i(r) \subseteq \alpha(r)$,

(ii) firing all rules in $\eta_i(r)$ enables r, and

(iii) no proper subset of $\eta_i(r)$ enables r.

then each $\eta_i(r)$ is called an enabling set of rule r. In other words, if rules in $\eta_i(r)$ fire, then r becomes enabled.

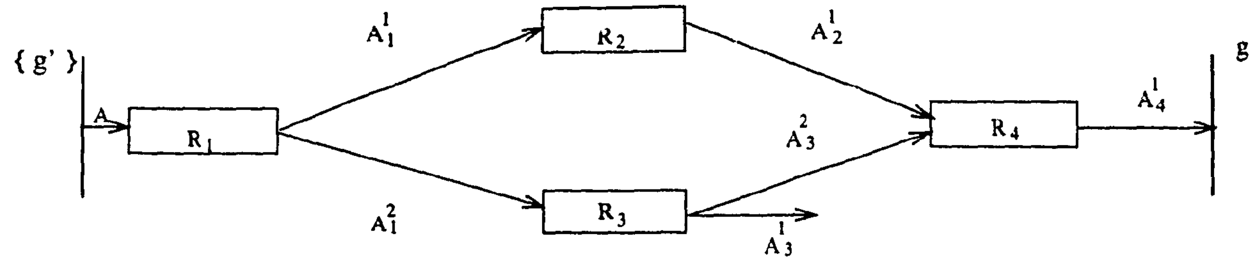
A path represents a connector, and all rules in a path should fire, by definition, once the goals required by the path are inferred. Thus, for every rule in a path, its enabling set is also contained in that path.

As an example, in Figure 4.1, the enabling set of rule R4 is { R2, R3 }, but the enabling set of rule R1 is $\emptyset$. But, this does not mean that the antecedent of rule R1 is empty (that is, does not contain any atoms). Since the rule accessibility relation is limited to the rule dependencies due to the unification of non-goal atoms, there must be at least one rule in a path that must be enabled due to previously inferred goals (this includes permissible initial evidence input by the user). These rules are called the **head rules** of a path. The atoms in the antecedent of a head rule in a path would not unify with any atom in the consequent of other rules in the path. Similarly, the rules that infer only goal atoms in a path are called **toe rules**. The atoms in the consequent of a toe rule would not unify with any atom in the antecedent of other rules in the path. It should be noted that a toe rule need not be distinct from a head rule: that is, a path can contain a single rule. Rules that are neither head rules, nor toe rules of a path are called the **body rules** of the path. In Figure 4.1, rule R1 is a head rule, rule R4 is a toe rule, and rules R2 and R3 are body rules.

Corollary 4 *Every path has at least one head rule, and one toe rule.*

This corollary is used by **path hunter**, a tool for extracting paths from a given rule base, discussed in section 4.3.1.

However, it should be noted that paths capture only potential rule execution sequences because it is the inference strategy that determines if a rule that is enabled can indeed fire. Further, the paths in a rule base is only a subset of the set of rule execution sequences in the rule base. But, as a goal specification is intended to abstract problem solving in a domain and paths in a rule base capture those rule sequences (hence, rule interactions) that infer goals, the analysis based upon paths and goals can provide useful insights towards understanding the working of a system [Grossner et al., 1993; Preece, Grossner, Gokulchander, and Radhakrishnan, 1993b; Preece et al., 1993a, 1993a]. The use of paths and goals as a basis for system evaluation to detect errors and anomalies in a system is described in section 4.2.



where the goal g = A_4^1 and G = { g' }

Figure 4.3: *A rule graph depicting a path that is unshaved since rule R_3 is not fully consumed.*

Two types of paths are possible in a rule base. Paths in which every non-goal atom inferred by a rule unifies with an atom in the antecedent of one other rule in the path are called **shaved paths.** The rules in such paths are said to be **fully consumed**. A path is said to be **unshaved** iff it is not a shaved path: that is, it contains at least one rule that is not fully consumed.

Definition 13 (Consumption of a rule) *A rule r in a path Φ from goal(s) $G = \{g_{i_1}, g_{i_2}, \dots g_{i_n}\}$ to g is fully consumed iff every non-goal atom in the consequent of r unifies with an atom in the antecedent of one other rule r' in the same path. A rule is partly consumed iff it is not fully consumed.*

As an example, in Figure 4.1 every rule is fully consumed, whereas in Figure 4.3 rule R_3 is partly consumed because the non-goal atom A_3^1 inferred by R_3 does not unify with any other atom in the antecedent of any other rule in this path. The notion of rule consumption in a path is useful for identifying some rule base anomalies (section 4.5).

By definition, a path captures the rule interactions that occur in a goal-to-goal progression because a path represents the transitive closure of the rule accessibility relation in that goal-to-goal progression. The extent to which a given rule base represents the acquired knowledge of goal inference is reflected by the paths in the rule base; they are collectively said to portray the **structure** of the rule base [Grossner et al., 1993]. Our model of a rule base is the set of paths inferring goals adhering to a design scheme.

Definition 14 (Rule Base Structure) *The structure of a rule base (or, simply structure) is defined as $< \mathcal{G}, \Pi, \mathcal{D} >$ where $\mathcal{G}$ is the goal specification of the domain, Π is a set of rule base paths, and $\mathcal{D} =< \mu_1, \mu_2 >$ is the adhered design scheme, such that,*

$$(1) \quad (\forall \Phi \in \Pi)\ (\exists G, g)\ (G \subset \mathcal{G})\ (g \in \mathcal{G})\ G \wedge \Phi \vdash g \quad \text{and}$$

$$(2) \quad (\forall g \in \mathcal{G})\ H \supseteq \begin{cases} \mu_1(g) & \text{if } g \text{ is a final goal} \\ \mu_2(g) & \text{if } g \text{ is an intermediate goal} \end{cases}$$

where H is the set of all hypotheses in the rule base.

The second condition in the above definition simply asserts that goals are inferred using rule base hypotheses only. Thus, if any external actions are to be modeled as part of a goal inference, it should still be represented using a hypothesis in the rule base; the action can take place following the inference of this hypothesis. During system evaluation, this can ensure that a given rule fires correctly by examining the inferred goal. This is particularly useful, when simulating external actions (that could take place during field operation) as part of a rule can be costly, or cannot be done during development. As an example, consider a life support system that monitors patient breathing, and turns on additional oxygen when oxygen intake falls below a threshold. In the system implementation, a rule should fire when the oxygen intake falls below a threshold, and turn on the appropriate oxygen equipment. In our model, this action should be represented using an hypothesis in the rule consequent in addition to executing the appropriate action. During system development, the rule firing can then be mapped to a path which can be inspected to check if the rule fires under the correct conditions. Note, the development site may or may not have the associated control equipment in this case owing to its cost.

The progression of problem solving at the structural level is then captured by a sequence of paths from permissible combinations of initial evidence (level-0 goals) to a solution (final goal). Such a connected sequence of paths is called a **route**. A route from a level-0 goal to a solution is called a **relevant route**; any other route from a level-n goal to a goal g', where $n \geq 1$ and g' is not a solution, is called an **irrelevant route**.

Definition 15 (Relevant and Irrelevant Routes) *A route is a connected sequence of one or more paths of the form $\Phi_j\Phi_{j+1}\ldots\Phi_n$ such that the goal inferred by path Φ_i is required for path Φ_{i+1}, where $j \leq i < n$. If Φ_j infers a level-1 goal and Φ_n infers a final goal, then the route is said to be a* relevant route; *otherwise, the route is said to be* irrelevant.

The importance of relevant and irrelevant routes would be apparent when performance evaluation aspects of a system is discussed (section 4.4).

The extraction of paths from a rule base using a given goal specification is a non-trivial problem because procedures that extract inference chains from a rule base have an exponential complexity in the worst case [Ginsberg, 1988; Kiper, 1992]. However, goal specification can be used to control the computation required to extract paths [Grossner et al., 1993] as will be apparent later. The extraction of paths from a rule base is termed **structure extraction**, and it influences a variety of evaluation processes for rule-based systems [Chander et al., 1994; Chang et al., 1990; Ginsberg, 1988; Kiper, 1992; Chander et al., 1995; Preece et al., 1994].

As a final note, chapter 1 (section 1.3) presented a development perspective for rule-based systems using three stages: the functional requirements stage (chapter 2, sections 2.1 and 2.2), the design stage (chapter 3, sections 3.1, 3.3, and 3.4), and the implementation stage (section 4.1). The next step is in explicating the reverse links between the three stages as shown in Figure 1.2 (page 15): relating how paths inferring goals (thus, mapping to the design stage), map to overall problem solving as viewed in the functional requirements stage. Every path in a rule base adhering to a design scheme maps to one of the goals designed as a conjunction of atoms in the design stage. This portrays the reverse link from the implementation stage to the design stage. The routes extracted from a rule base portray how a goal graph is actually traversed in the system implementation at run time. This progression of problem solving, abstracted in terms of routes, allows one to appreciate how goals designed during the design stage and inferred in the implementation stage map to the functional requirements stage of the system, where the goals were conceived. This explicates the reverse link from the design stage to the functional requirements stage.

4.2 Evaluating Rule-based Systems: Preliminaries

Systematic evaluation methodologies, tools, and techniques for rule-based systems are important in order to assess their reliability, verifiability, and other qualities [Preece, 1990; Giovanni, 1989; Preece et al., 1993a]. System qualities manifest in two forms: internal and external [Ghezzi et al., 1991]. The external qualities are those that are visible to the system users (for example, run-time performance), whereas the internal qualities are those that are visible to system developers (such as maintainability). The internal qualities are required to achieve some external qualities: for example, the internal quality [illegible]erifiability is required to achieve the external quality of reliability [Preece et al., 1993a].

For ease in understanding the evaluation methods discussed, we use a case study of a rule-based system to describe a university environment. The body of knowledge acquired by interviews for this domain is shown in Figure 4.4; the goal specification of the domain is shown in Figure 4.5; and the inviolables are shown in Figure 4.6. Note, permissible combinations of initial evidence (which represent conjunction of atoms representing initial evidence that a domain expert would use to start problem solving) should also be specified. In order to show the effect of each initial evidence on problem solving, each atom that is an initial evidence is shown as a level-0 goal. Thus, permissible combinations of initial evidence would be represented as AND edges from level-0 goals in the goal graph. Suppose design scheme <F1, I3> is chosen for this domain, a rule base to encode the goal progression knowledge is shown in Figure 4.7.

In the following sections, a set of evaluation techniques for rule-based systems are outlined. We also show the use of paths and goals in these techniques.

4.3 Structural Validation of Rule-based Systems

The process of validation ensures that the system conforms to its requirements. In general, the validation process for a system is started after defining a validation

Knowledge Acquired Via Interviews

" All undergraduates hold unique green bordered ID cards."
"Most undergraduates here hold a GPA that is higher than national average."
"Though they are hard working, generally no financial aid is available."
"A hard working student is likely to take honors courses."
"All undergraduates are considered as regular academic students who are enrolled using a registration process."
"Good grades in junior college is required for admission into university."
"Only regular academic students can enroll during registration."
"All registered undergraduates are young."
"Undergraduate students do not receive financial aid; however, students with good record and good grades in junior college can expect bursaries on a competitive basis making their life comfortable."
"Only undergraduates can register for honors courses and those in the dean's list (with high GPA, above 3.5) and taking honors courses can pass with distinction; such high GPA requires hard work from the undergraduates."
"University graduates with distinction have a good career in industry and a comfortable life."

Figure 4.4: *Typical knowledge elicited from a domain expert about a university environment. Interviews are the most common ways of eliciting knowledge from the domain. These are later translated into a set of goals associated with the domain by the knowledge engineer in collaboration with the domain expert forming the goal specification for the domain.*

The Goal Specification.
$\text{REGISTERED}(x) \wedge \text{GREENBORDRID}(x)$
$\text{GREENBORDRID}(x) \wedge \text{HARDWORKING}(x)$
$\text{REGISTERED}(x) \wedge \text{DEANSLIST}(x)$
$\text{ACADEMIC}(x)$
$\text{GOODCAREER}(x)$
$\text{BURSARY}(x)$
$\text{COMFLIFE}(x)^*$
$\text{ACADEMIC}(x) \wedge \text{YOUNG}(x)^*$
$\text{GOODGRADES}(x,y) \wedge \text{GT}(x, Gpa, 3.5)^*$

Atoms denoting initial evidence are as follows:
$g_{01} = \text{REGISTERED}(x)$
$g_{02} = \text{GREENBORDRID}(x)$
$g_{03} = \text{HARDWORKING}(x)$ and
$g_{04} = \text{DEANSLIST}(x)$

Figure 4.5: *The goal specification of the university domain from its description in Figure 4.4. An asterisk on a goal indicates that it is a final goal.*

$$\text{GRAD}(x) \wedge \text{UGRAD}(x)$$
$$\text{NOFINAID}(x) \wedge \text{BURSARY}(x)$$

Figure 4.6: *The inviolables of the university domain. It is assumed that a registered student cannot be both undergraduate and graduate. In addition, if a person receives a bursary, he/she cannot be categorized as not receiving any financial aid.*

$$\begin{aligned}
R1 &: \text{REGISTERED}(x) \wedge \text{GREENBORDRID}(x) \rightarrow \\
&\quad \text{ENROLLED}(x) \wedge \text{NOFINAID}(x) \\
R2 &: \text{ENROLLED}(x) \rightarrow \text{ACADEMIC}(x) \wedge \text{STUDENT}(x) \\
R3 &: \text{GREENBORDRID}(x) \wedge \text{NOFINAID}(x) \rightarrow \text{UGRAD}(x) \\
R4 &: \text{STUDENT}(x) \wedge \text{UGRAD}(x) \rightarrow \text{ACADEMIC}(x) \\
R5 &: \text{GREENBORDRID}(x) \rightarrow \text{NOTGRAD}(x) \\
R6 &: \text{REGISTERED}(x) \wedge \text{NOTGRAD}(x) \rightarrow \\
&\quad \text{STUDENT}(x) \wedge \text{ACADEMIC}(x) \wedge \text{YOUNG}(x) \wedge \text{UGRAD}(x) \\
R7 &: \text{DEANSLIST}(x) \rightarrow \text{HIGHGPA}(x) \wedge \text{HONSCOURSES}(x) \\
R8 &: \text{REGISTERED}(x) \wedge \text{HONSCOURSES}(x) \rightarrow \text{UGRAD}(x) \\
R9 &: \text{UGRAD}(x) \wedge \text{DEANSLIST}(x) \rightarrow \\
&\quad \text{GOODGRADES}(x, Juniorcollege) \wedge \text{COMPLETED}(x, JuniorCollege) \\
R10 &: \text{HIGHGPA}(x) \wedge \text{REGISTERED}(x) \rightarrow \text{GT}(x, Gpa, 3.5) \\
R11 &: \text{HARDWORKING}(x) \rightarrow \text{HONSCOURSES}(x) \\
R12 &: \text{GREENBORDRID}(x) \rightarrow \text{HIGHGPA}(x) \\
R13 &: \text{HIGHGPA}(x) \wedge \text{HONSCOURSES}(x) \rightarrow \text{UGRAD}(x) \wedge \text{DISTINCTION}(x) \\
R14 &: \text{UGRAD}(x) \wedge \text{DISTINCTION}(x) \rightarrow \text{GOODCAREER}(x) \\
R15 &: \text{GOODRECORD}(x, Juniorcollege) \wedge \text{COMPLETED}(x, Juniorcollege) \\
&\quad \rightarrow \text{GOODCAREER}(x) \wedge \text{BURSARY}(x) \\
R16 &: \text{GOODCAREER}(x) \rightarrow \text{COMFLIFE}(x) \\
R17 &: \text{BURSARY}(x) \rightarrow \text{COMFLIFE}(x)
\end{aligned}$$

Figure 4.7: *The rule base encoding the knowledge describing the university domain.*

criteria [O'Keefe et al., 1987]. A typical set of criteria are as follows [Preece, 1992]:

(a) What to validate?

(b) What tests are to be conducted?

(c) What is the standard against which validation will be performed?

Note that we have deliberately omitted some aspects like "who will validate?", "when to validate?", etc. These questions are relatively minor for a laboratory validation (validation performed under controlled conditions; tests and results are usually reproducible), but not for a field validation (the system validation at its site).

Structural validation is conducted in order to ensure that the structural subcomponents of a system are indeed correct, and contribute to the correct execution of its function [Preece, 1992; Ghezzi et al., 1991]. However, the process of structurally validating a rule-based system is non-trivial because of the non-sequentiality of rule firings and rule interactions. An explicit model to capture rule interactions that occur during problem solving is required. The popular approach among the researchers is to capture the rule interactions by defining paths using a formalism, and extracting the paths from a rule base to inspect whether the rule interactions captured via paths are indeed the interactions desired by a system designer [Kiper, 1992; Chang et al., 1990; Rushby, 1988]. However, the notion of path in a rule base among researchers is not unique.

For structurally validating a rule-based system, the implementation model discussed in section 4.1 can be used. This way of structurally validating rule-based systems has several advantages compared to the other approaches [Kiper, 1992; Chang et al., 1990; Rushby, 1988]:

(a) Every path infers a goal, thus it makes it possible to map a set of rules to a definite aspect of problem solving: towards solving a sub-problem. This brings a notion of meaningfulness to the purpose of firing a set of rules.

(b) Every path is self-contained: once the goals required by a path are inferred, the path can fire. Thus, rule interactions that occur for a given goal-to-goal progression are completely localized and captured in a path.

(c) The formal characterization of a path in this model can act as a specification for automatic path enumeration by a tool [Gokulchander, Preece, and Grossner, 1992].

(d) Use of goal specification can be used for controlling the combinatorial explosion that occurs when trying to enumerate paths [Grossner et al., 1993].

(e) Goals and paths can also be used for other evaluation procedures; thus, paths and goals provide a common basis for system evaluation [Preece et al., 1993a, 1994; Chander et al., 1994, 1995, 1995b, 1995c].

The structural validation of a rule-based system consists of two steps:

1. extracting the paths from a rule base; and
2. measuring the extent to which the set of paths in a rule base are exercised by a test suite (called **path coverage**).

A test suite consists of a set of test cases (a set of atoms denoting initial evidence). To measure the path coverage, every test case in a test suite would be input to a system, and the extent paths are exercised would be observed. An ideal test suite is one that should achieve 100% path coverage, but is either difficult to obtain, or too costly to construct in practice [Preece, 1992].

To structurally validate the example rule base shown in Figure 4.7, the paths from the rule base must be extracted. They are shown in Figure 4.8.

The next step in validating this system would be to measure the extent paths are exercised for a test suite. However, we have to simulate a run time control that would typically be exerted by an inference engine. Often, the inference engine strategy can affect the measurement of path coverage because a path may not be covered fully during run time, yet its goal may be inferred. For example, this can happen if the inference engine adopts a strategy called "chilling of prod rules" [Shinghal, 1992]: that is, a rule would be prevented from firing if its consequent is already contained in the working memory. Thus, if the consequent of the head rules of a path is already contained in the working memory, then the head rules would not fire. But, the rest of the rules in the path can fire, when the goals required by the path are inferred.

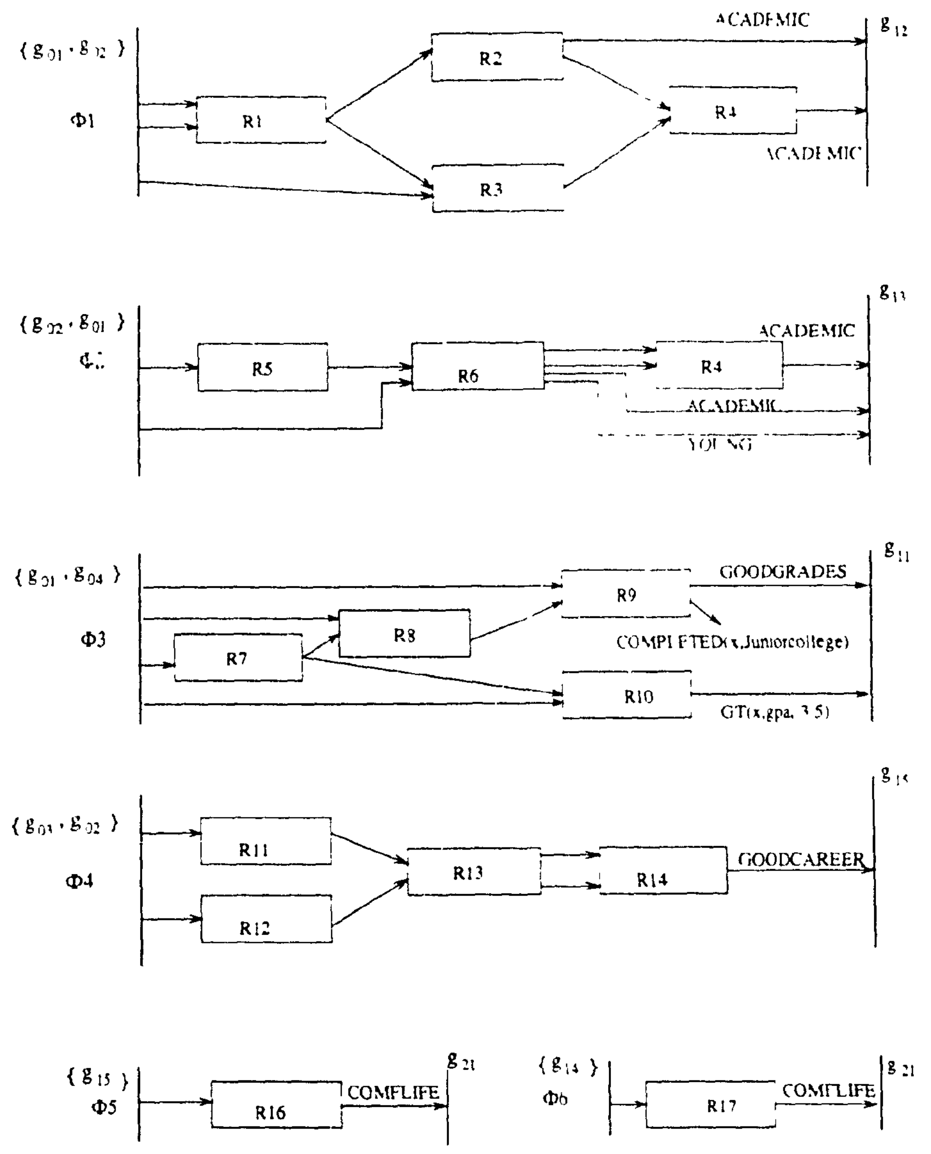


Figure 4.8: *Paths extracted from the rule base of Figure 4.7. To avoid cluttering, not all of the inferred atoms in a path are shown.*

Test	Path Coverage						Overall
case	Φ1	Φ2	Φ3	Φ4	Φ5	Φ6	coverage
1	0%	33.3%	0%	100%	100%	0%	35.3%
2	0%	0%	100%	50%	100%	0%	41.2%

Figure 4.9: *Summarizing the test coverage for the example system for a test suite containing two test cases.*

To illustrate path coverage measurement for the example system, we will adopt the following inference strategy: the order in which the enabled rules are fired is the same as their physical order in the rule base. To measure path coverage, we use the following criteria: if a rule appears in multiple paths, the rule firing is mapped to every path in which that rule appears. For our example rule base in Figure 4.7, assume that a test suite T containing two test cases (1) {`HARDWORKING(Henry), GREENBORDRID(Henry)`}, and (2) {`REGISTERED(Henry), DEANSLIST(Henry)`} was applied.

The path coverage for the first test case in T would be as follows:

- Path Φ2 would be partially covered: rule R5 would fire in path Φ2 (but no goal would be inferred). Thus, path Φ2 is considered covered 33.3% (1 rule out of 3 fired).
- Paths Φ4 and Φ5 would be exercised fully (100% coverage).
- Paths Φ1, Φ3, and Φ6 would not fire at all (0% coverage).

Thus, the overall coverage for this test set would be 35.3% because 6 rules fired out of a total of 17 rules in the rule base.

On similar lines, if the second test case of T is applied, paths Φ1, Φ2, and Φ6 would be covered 0%, paths Φ3 and Φ5 would be covered 100%, and path Φ4 would be covered 50% (rules R13 and R14 fire). The overall coverage of the rule base would be 41.2% (7 out of 17 rules fired). The coverage of the example system for test suite T is summarized in Figure 4.9. This example rule base (see Figure 4.7) was coded using CLIPS, and the above test suite T was applied. The coverage results obtained concur with the analysis summarized in Figure 4.9, but the order in which the rules fired was different. This occurred because the CLIPS inference strategy does not guarantee that the rule execution order would match with their physical order in the rule base [Giarratano and Riley, 1993].

There are two main observations that are apparent from this example:

1. Even for a simple rule base, a 100% coverage of all the paths was not achieved despite the fact the test suite contained all the initial evidence. This emphasizes the need for a careful construction of test cases for a domain.

2. For the second test case, the coverage of path Φ4 deserves attention because according to the goal requirement of path Φ4, it should not have been covered. It will be apparent later that this occurred because of a coding error in the rule base causing rule subsumption (section 4.5).

This validation example also illustrates that the actual events that take place at run time need not map uniquely to a model because of its simplifying assumptions. We elaborate more on this in section 4.3.2.

We developed two tools to facilitate structural validation of rule-based systems: the tool **path hunter** extracts paths from a rule base using a given goal specification [Grossner et al., 1993], and the tool **path tracer** measures the extent paths in a rule base were exercised for a given test suite [Preece et al., 1993b]. The appropriate use of these tools is illustrated in Figure 4.10. **Path hunter** uses the rule base of the system along with the goals specified in a declarations file to extract the paths from the rule base. It is described in section 4.3.1. **Path tracer** works with the paths extracted from a rule base and a set of run traces of the system (files capturing rules fired and hypotheses inferred for a test run), and provides measures to reflect the extent paths are exercised by a given test suite. It is described in section 4.3.2.

Though the method of structural validation that we discussed is the same in principle for any rule-based system (that is, independent of the size of a system), the pragmatic issues involved in path extraction, and in measuring path coverage for large rule bases deserve special attention. Further, expert system shells like CLIPS allow procedural constructs like `if, while,...` on the consequent of a rule for coding convenience [Giarratano and Riley, 1993]. Rules containing such procedural constructs,

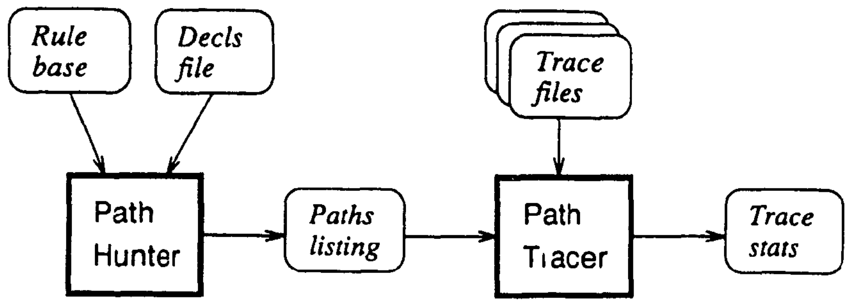


Figure 4.10: *Using the* path hunter *and* path tracer *tools for structurally validating a rule-based system.*

strictly speaking, are not declarative. In addition, as many rule-based systems currently exist, the goal specification has to be reverse engineered from the existing rule base for validation, and needs to be refined to control the computation for path extraction. Thus, the question arises: how should the above method be modified for such systems? These details are outlined in the following sub-sections. As a final note, the path statistics obtained from a rule base as part of system validation also provides insights into certain qualities of the rule base (complexity, verifiability, etc). This aspect is described in section 4.6.

4.3.1 Path Hunter: A Tool to Extract Paths from a Rule Base

Path hunter has several modules that are needed when extracting paths from a "real life" rule base where rules are not guaranteed to be declarative as assumed in this framework (section 3.1). This requires that such rules be pre-processed into a set of declarative rules before applying our implementation model to extract the paths from a rule base. The various modules that comprise the path hunter tool, its design, and its implementation are only briefly explained below. For additional details, refer to the technical report [Gokulchander et al., 1992].

The first step in path extraction is to to obtain the items on the antecedent and the consequent of every rule. In our case, the rule base was coded under CLIPS expert system shell [Giarratano and Riley, 1993]. The CLIPS tokenizer[1] was used to

[1]This CLIPS parser was written by A. Preece as part of his COVER [Preece et al., 1992b]

```
(defrule R-i
   (P)
=>
   (assert (Q))
   (if (cond) then
      (assert (R)))
) ; end of rule R-i
```

Figure 4.11: *An example rule adhering to the syntax of the CLIPS expert system shell.*

parse a CLIPS rule base and obtain the items on the antecedent and the consequent of each rule as separate tokens. Foι example, the tokens obtained correspond to the constructs like `defrule, assert, if-then-else, while,` ... that can be present in a rule coded using the syntax of CLIPS. It should be noted that the CLIPS tokenizer is not actually a part of path hunter, but is only required to pass the various constructs of a rule coded using the syntax of CLIPS as tokens to path hunter for processing. Path hunter is independent of the expert system shell used to develop the rule base as long as a parser is available to parse the rule constructs and provide a sequence of tokens for each rule (consisting of its name, antecedent, and consequent) to path hunter.

From the set of tokens obtained, every rule in the rule base is mapped into a rule of the form: *Antecedent* $\rightarrow$ *Consequent*, as viewed in the design stage. The module that maps a CLIPS rule into a set of declarative rules of the above form is called the **rule splitter** as its function appears to "split" a given CLIPS rule and re-write it as a set of declarative rules. This is necessitated because CLIPS allows procedural constructs in the consequent of a rule, user defined functions, and additional operators for flexibility in development. In CLIPS, it is perfectly legal to have a rule to contain procedural constructs such as `if` ... `then` ... `else, while`, etc in its consequent. For example, the rule

$$R_1 : P \rightarrow Q, if\ (cond)\ R.$$

can be encoded in CLIPS as shown in Figure 4.11.

Whenever a rule encodes a (procedural) conditional construct such as an `if` in

verification tool.

its consequent, two situations can be observed when the rule fires. For example, two situations are possible when the rule R_t shown in Figure 4.11 fires depending upon the boolean condition *cond*. If this condition is false, then the rule infers Q; otherwise, it infers Q and R. But, this rule is not in the form that is stipulated in the design stage. Rule splitter cuts a rule containing a conditional construct in its consequent into two rules: one rule corresponds to the situation when the boolean condition in the construct fails, and the other when it succeeds. More specifically, the CLIPS rule R_t in Figure 4.11 would be split into two rules: a rule that infers only Q, and a rule that infers Q and R.

In splitting rules, however, we need to preserve the effect of external function calls, if any, in a conditional construct. When an external function appears in a rule, it is modeled as a predicate on the antecedent, or consequent (as appropriate) of the split rule generated by this module. Thus, for a rule base containing external function calls, information pertaining to such external functions should also be supplied to path hunter.

Each rule that is split by the rule splitter is registered and the following information is captured:

1. the split name: usually of form *OriginalName%n*, where *OriginalName* is the original name of the rule;
2. the antecedent of the rule as a set of predicates used in the antecedent of the original rule, and those predicates representing external functions;
3. a set of atoms for the rule consequent modeling a possible truth value for the boolean conditions in the rule consequent (if any);
4. provision is made for retractions, but is left currently empty as we do not handle negated atoms in this formalism; and
5. the original name of the rule to maintain the origin of the split rule.

Sometimes, several split rules are identical due to the abstractions in treating conditionals and external functions. The set of split rules that are identical are said to form a **rule equivalence class**. A rule equivalence class is treated as a single rule by **path hunter** for reasons that will be apparent later.

The presence of a module in **path hunter** to handle procedural constructs in a rule does not imply that a programmer can use these constructs without care. Though CLIPS provides constructs such as `if`, `while`,..., for convenience in encoding, their use is discouraged in general [Giarratano and Riley, 1993]. Further, rule base analysis becomes cumbersome when individual rules mix procedural and declarative constructs [Rushby, 1988]. More specifically, for rule base analysis using **path hunter**, heavy use of procedural constructs on the consequent of a rule can bring down the efficiency of rule splitting, and hence, that of path extraction owing to the following result.

Lemma 3 *The rule splitting algorithm is of exponential complexity in the worst case.*

Proof. *Consider a rule R that has n conditional constructs in its consequent as shown below.*

```
(defrule R
   (P)
    ⋮
=>
   (if ($cond_1$) then
      (assert (R1)))
   (if ($cond_2$) then
      (assert (R2)))
      ⋮
   (if ($cond_n$) then
      (assert (Rn)))
) ; end of rule R
```

Every conditional $cond_i$ can be true or false. Thus, when the above rule R fires there are 2^n possible situations each inferring a combination of the atoms $R1, \ldots, Rn$ in the consequent of rule R. Thus, 2^n split rules should be generated in order to account for these 2^n situations.

However, in practice the value of n, the number of conditional constructs in a rule consequent, is usually small (at most 2 or 3), and rule splitting does not impose a major overhead. The above lemma serves as a caution to developers "extensive use of procedural constructs in encoding knowledge (perhaps done by the designer to minimize the total number of rules) would make the later analysis and evaluation of the rule base cumbersome."

Once the rules in a rule base are represented in a declarative form, we are concerned with the extraction of the accessibility relation between the rules (definition 11, page 79) to determine the enabling sets for each rule (definition 12, page 81). The accessibility between the rules is determined from the output of the rule splitter by checking if a non-goal atom inferred by a rule r_1 unifies with an atom in the antecedent of another rule r_2. The rule accessibility that is captured is then used to determine the enabling sets for each rule. Finally, the tool obtains rule chains that start with one or more head rules, and terminates on a toe rule because every path starts (ends) with at least one head (toe) rule by corollary 4 (section 4.1). This is done by obtaining the enabling sets of toe rules, and repeating this step (recursively) for every rule contained in the enabling sets obtained until one or more head rules are encountered. Every such rule chain containing a toe rule and one or more head rules is called a **fragment**. Finally, the set of fragments are checked for goal inference to identify the rule base paths. The paths produced by **path hunter** for the example rule base is shown in Figure 4.12.

A major issue in extracting paths from a large rule base is the number combinations in which a given rule can be enabled. For small rule bases, **path hunter** executed efficiently and extracted the paths, but when the size of the rule base is increased, the number of enabling sets for individual rules increased considerably. Since **path hunter** recursively enumerates these dependencies, the number of combinations to be enumerated increased so rapidly, a *combinatorial explosion* was encountered (causing the tool to run for hours without terminating). Though this computation can be controlled partly by restricting the tool to extract only shaved paths from a rule base, its run-time performance required optimization when handling large rule bases in general [Grossner et al., 1993].

Path Hunter Output	Path in Figure 4.8 (page 91)
`path(example,['R1%1','R2%1','R3%1','R4%1']).`	$\Phi 1$
`path(example,['R4%1','R5%1','R6%1']).`	$\Phi 2$
`path(example,['R10%1','R7%1','R8%1','R9%1']).`	$\Phi 3$
`path(example,['R11%1','R12%1','R13%1','R14%1']).`	$\Phi 4$
`path(example,['R16%1']).`	$\Phi 5$
`path(example,['R17%1'])`	$\Phi 6$

Figure 4.12: *Output of* path hunter *for our example rule base shown in Figure 4.7. The* path *predicate above also contains the accessibility of the rules in the path, but is not shown for clarity. In the above output, the argument* example *in the* path *predicate refers to the task name of the path. Note that the paths are enumerated in terms of split rules.*

The combinatorial explosion in a rule base arises primarily due to the following three reasons [Grossner et al., 1993; Preece et al., 1994]:

Case 1. when several split rules are identical;

Case 2. when a specified goal is too general; and

Case 3. the multitude of rule dependencies arising out of non-goal atom(s).

When several split rules are identical, the number of combinations to be enumerated increases since every rule that is identical to a rule r can replace r in every rule sequence in which r can appear. Thus, for handling case (1), path hunter uses a rule equivalence class to replace identical split rules.

When a goal is too general, then too many rule sequences infer that goal; this increases the number of combinations to be handled when enumerating paths inferring that goal. To handle case (2), the goal is specialized: goal specification is refined so that a goal that is too general is either made more specific, or split into a number of specific goals.

Whenever a combinatorial explosion was encountered due to case 3 above, the rule dependencies were checked to see the cause for this combinatorial explosion. Suppose m rules infer a non-goal atom A that unifies with the antecedent of n other rules, then there are $m \times n$ dependencies between the rules that are created; this number $m \times n$ can be treated as indicative of the "importance" of the atom A in terms of the knowledge it encodes. In order to control the computation due to these rule dependencies, the acquired knowledge and the goal specification are checked to see if the knowledge represented by atom A should be incorporated into a goal, thus making A a goal atom. In the event where this is not possible one can augment the goal specification by *goal incorporation*: a new goal is introduced that uses the knowledge encoded in A. The justification for this is as follows: if several rules infer an atom and many others use it in their antecedent, clearly this atom is important for problem solving; if not, it indicates a possible error in rule encoding and the rules need to be modified. Thus, inferring atom A can be related to achieving an important state, or milepost in problem solving. As the purpose of goal specification is to identify these states as completely as possible, it is refined to include atom A. For practical purposes, these heuristics are usually sufficient to control the combinatorial explosion, if it arises during path extraction [Grossner et al., 1993].

Overall, goal specification can be very handy in controlling the computation for path extraction because it can cut down the rule dependencies whenever they cause a combinatorial explosion. However, abusing this flexibility (for example, by making changes to goal specification in an ad hoc manner solely to control this computation) can undermine the usefulness of goals to abstract problem solving in general (section 4.4). This further emphasizes the importance of design. As a final note, path hunter requires every specified goal to be associated with a task. For example, the goal graph of the domain shown in Figure 2.3 (page 21) can be characterized using two tasks: a biliary diagnosis task, and a liver diagnosis task. Grouping goals by tasks can further facilitate validation by providing selective control over a portion of a rule base (section 4.3.3).

4.3.2 Path Tracer: A Tool to Measure Path Coverage

The purpose of a path tracer is to measure the extent to which paths are exercised for a chosen set of test cases [Preece et al., 1993b], that is, it measures path coverage. This allows us to observe the following run time characteristics of a system:

1. the frequency with which a rule participates in problem solving for goal inference (a measure of rule activity); and
2. the number and the extent paths were exercised for the given test suite (a relative measure of system performance).

In addition, the path characteristics output by the tool helps to portray how realistic is our model for analyzing the run time performance of a system in terms of its abstractions, and can help in model refinement, if necessary. Though the results of running **path tracer** is collected over a set of test cases applied to the system, it does not imply that this set of test cases is representative [Preece and Shinghal, 1992; O'Keefe et al., 1987]. **Path tracer** simply provides some system statistics observed at run time and measured in terms of the number of paths exercised and their characteristics. Of course, the results can be utilized for tuning the performance of the system.

Unfortunately, mapping a set of rules fired at run time into a set of paths is a non-trivial task. For example, a CLIPS trace file only records the rules fired in that sequence and their inferred hypotheses [Giarratano and Riley, 1993]. The first step in path tracing is to map an observed rule firing in a trace file into the rule abstractions used by **path hunter** (because the paths are enumerated in terms of split rules). Measuring path coverage for systems coded using an expert system shell is, however, complicated because the rules observed in a run trace need not map one-to-one to the rule abstractions made by **path hunter** when extracting paths. More specifically, an observed rule firing can match more than one split rule. Such a rule mapping is said to be **equivocal** because it cannot uniquely identify a rule as part of some path.

Path tracer uses the notion of a **thread** in mapping an observed sequence of rules to a path. A thread is a linear sequence of rules with an head rule and a toe rule. For example, in Figure 4.13 there are two threads: D1 and (D2, D3), where the D's label the rule accessibilities in a thread. A path is analyzed in terms of the threads it contains. In so doing, **path tracer** measures path coverage in terms of the rule accessibilities present in a thread using three counting strategies. The three strategies vary in the extent they count the accessibility between two rules in a thread as part

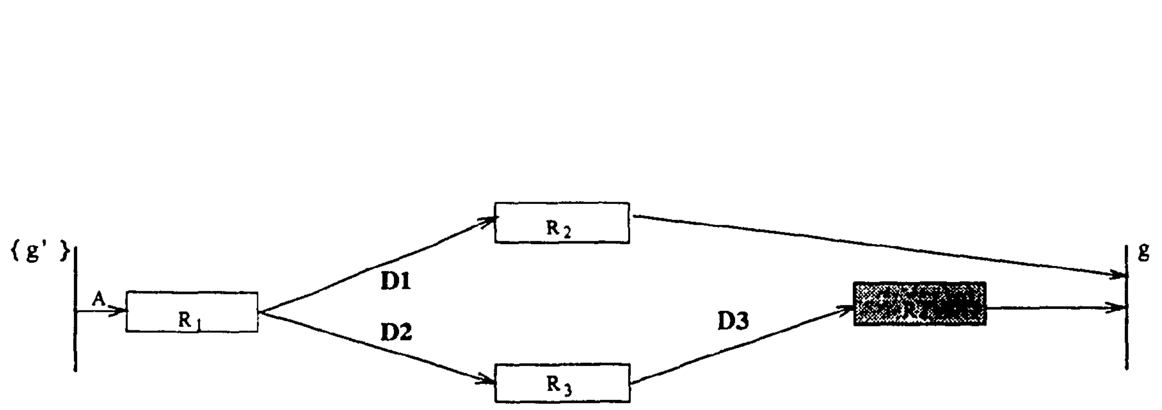


Figure 4.13: *Illustrating threads in a path used for measuring path coverage from run trace information. The rule shown in the greyed box is assumed to be involved in an equivocal mapping.*

of path coverage, when one of the rules is involved an equivocal mapping. They are described below.

- The **Liberal** strategy: This takes into account the accessibility that is observed even if a rule cannot be uniquely mapped to a split rule. In other words, it allows for threads to have an equivocal mapping, but counts only the first accessibility arising because of the rules involved in a equivocal mapping. Referring to Figure 4.13, this would count all the accessibilities (D1, D2, D3) of the path as seen in the trace file.
- The **Moderate** strategy: It allows for the presence of rules which cannot be mapped uniquely to a split rule in a thread, but does not measure them as part of the thread. In other words, it allows for threads that have an equivocal mapping, but does not count the accessibility arising because of the rules involved in a equivocal mapping. Referring to Figure 4.13, this would count only two accessibilities (D1, D2) of the path as seen in the trace file.
- The **Conservative** strategy: It does not allow for any accessibilities arising due to the presence of rules that cannot be uniquely mapped to a split rule. In other words, this does not allow any thread of a path to have an equivocal mapping. Referring to Figure 4.13, this would count only one accessibility (D1) of the path as seen in the trace file.

Since path coverage is measured in terms of the rule dependencies in a trace file, it can be expressed as a percentage. If all the accessibilities in a path are counted in the trace file, then the path coverage is 100%. Otherwise, the number of accessibilities of a path that are counted can be used to determine the percentage coverage of that path. Thus, for the path in Figure 4.13, the coverage for the various strategies are as follows:

- based on the liberal strategy, this path coverage is 100% (because all the three accessibilities are counted);
- based on the moderate strategy, this path coverage is 66.7% (because only two accessibilities are counted); and
- based on the conservative strategy, this path coverage is 33.3% (because only one accessibility is counted).

The above measures provide a broad perspective in viewing path coverage in terms of the observed rule causalities in a trace file. The above measures also reflect that a model of a system abstracts events that take place in the system, thereby simplifying the assumptions imposed on it for ease in understanding the behavior of the system. However, in order to map the actual events that occur (at run time) into abstractions of a model, we need to allow for the simplifying assumptions that were made during model definition, and should rely on a set of measures associated rather than one absolute measure [Preece et al., 1993b].

4.3.3 Validating a Large Rule-based System: A Case Study

To experiment the feasibility of extracting paths adhering to our definitions (section 4.1), we chose an existing rule-based system called Blackbox expert because it solves a puzzle called Blackbox. The Blackbox puzzle is an abstract diagnosis problem [Simon, 1973]. This is a large system consisting of 442 rules developed using the CLIPS expert system shell [Giarratano and Riley, 1993; Preece et al., 1994]. The size of the rule base of this system and the complex way in which the rules encode knowledge makes this system an ideal test bed to check for the pragmatic aspects involved in extracting paths. A sample rule from the Blackbox expert rule base is shown in Figure 4.14. As the system was not developed using an initial goal specification, it had to be reverse engineered from the rule base, and refined to facilitate validation.

```
(defrule adjust-shot-value
    (declare (salience 200))
    (phase selection)
    ?var1 <- (ADJUST-SHOT ?rule-ID ?r ?c ?new-val ?code)
    ?var2 <- (SHOTLEFT ?r ?c ?level ?total ?count)
  =>
    (retract ?var1 ?var2)
    (bind ?new-total (+ ?total ?new-val))
    (bind ?new-count (+ ?count 1))
    (bind ?new-level (/ ?new-total ?new-count))
    (assert (SHOTLEFT ?r ?c ?new-level ?new-total ?new-count))
    (if (= ?code 0) then
       (assert (ADJUSTED-SHOT ?rule-ID ?r ?c)) )  ; end if.
)  ; end rule adjust-shot-value.
```

Figure 4.14: *A sample rule from the Blackbox expert's rule base.*

The Blackbox puzzle consists of an 10×10 opaque grid, called the grid map, hiding a certain number of balls. The task of the system is find the location of the balls. To do so, the system fires "beams" into the box. The beams get absorbed ("hit") if their trajectory is incident on a ball, or get deflected if their trajectory is adjacent to a ball. The hits and deflections are observed. Based on the deflections and the hits, the system *analyses* the existing situation to infer position of the balls. If it cannot determine the locations of all the balls, then it *selects* beams to be fired from certain locations so that they can provide additional analysis information. This process of beam selection and beam analysis continues until all balls are located, or the system cannot identify some balls. In addition, the system is expected to minimize the number of beams fired while trying to identify ball locations. The system was developed independently by a research group who have also functionally validated the system to compare its ability with a set of humans on selected set of test cases. The problem solving ability of the system relative to the human subjects was found to be acceptable [Grossner, Lyons, and Radhakrishnan, 1991].

Due to the large size of the rule base of the Blackbox expert, a combinatorial explosion was encountered during path extraction. This combinatorial explosion was controlled using rule equivalence classes, goal specialization, and by augmenting the

Strategy	0%	> 0%	> 50%	> 60%	> 70%	> 80%	> 90%	100%
Conservative	35.9	64.2	32.4	26.2	20.0	18.6	17.6	17.4
Moderate	28.3	71.7	50.6	46.9	36.4	27.5	19.0	18.4
Liberal	28.3	71.6	52.8	50.1	45.1	42.2	35.0	33.3

Figure 4.15: *Path coverage of the Blackbox expert for a test suite containing 17 test cases.*

goal specification by goal incorporation (section 4.3.1). Goal grouping by tasks required by **path hunter** made it easier to focus on the portion of the rule base where combinatorial explosion was encountered, and thus allowed us to refine only a subset of rules and goals.

Results: The CLIPS tokenizer was used for parsing the rule base of the Blackbox expert. The execution of **path hunter** on the Blackbox expert's rule base containing 442 CLIPS rules produced 512 split rules (rule abstractions that adhere to our model), 72 equivalence classes, and resulted in the identification of 516 paths [Grossner et al., 1993]. These paths were inspected by the system designer and were certified to portray the desired rule interactions in inferring goals. Rule splitting also identified several rule coding errors by identifying several "undesirable" split rules to inspect and fix the original rule that was the source of the split rules.

The path coverage of the Blackbox expert for a test suite containing seventeen test cases is shown in Figure 4.15.[2] The numbers in Figure 4.15 must be interpreted as follows: for example, a number of 20.0 in the conservative strategy under column "> 70%" is interpreted to mean that for the given set of test cases, 20 percent of the 516 paths had a coverage of more than 70%. Similarly, 33.3% of the paths had 100% coverage using the liberal strategy for the given test set.

In addition, path tracing also provided insights into the activity of the various rules in a path. Rule activity of a rule r refers to the relative frequency of firings of this rule compared with the other rules. In particular, we are interested in the head rules of a path because they are the first rule(s) in a path to fire. Thus, head rule activity indicates which paths at least begin to fire for a given set of test cases, and

[2]Data reproduced (after correction) from [Preece et al., 1993b].

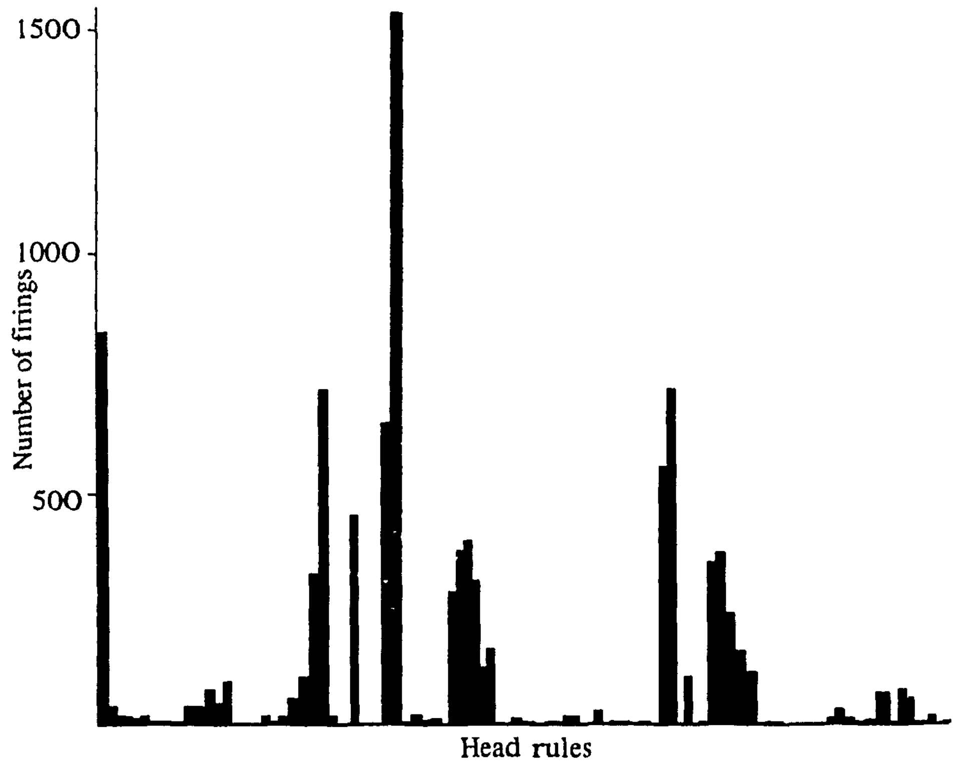


Figure 4.16: *Activity of the Blackbox expert's head rules.*

hence, the overall extent a path participates in problem solving (that is, it indicates the path firing frequency [Preece et al., 1993b]). This information can be used for system optimization, if necessary. For example, rule activity of head rules in Blackbox expert's rule base is shown using an histogram in Figure 4.16.

Though the results of (previous) testing of the Blackbox expert using the same test suite was considered satisfactory [Grossner et al., 1991], the structural validation of the system revealed that a significant portion of the rule base was never exercised for this set of test cases. This indicated that a large part of the rule base may need to be refined (see Figure 4.15). It also allowed a better understanding of the blackbox expert's problem solving and the interactions that occur between the rules. Path enumeration helped to uncover some coding errors in the rule base: for example, coding errors were revealed when some rules did not appear in any path. The structural validation experiment based on our model resulted in a better understanding of the system, and also demonstrated the applicability of this model for analyzing large rule bases (using **path hunter** and **path tracer**). This was one of the primary objectives behind developing the implementation model discussed in section 4.1.

4.4 Performance Evaluation of Rule-based Systems

The term performance is used frequently to denote different qualities: extent of anomalies, efficiency, etc. A survey of current works emphasizes the need for precise definitions to quantify the notion of performance [Guida and Mauri, 1993; Hamilton et al., 1991] because of the different interpretations used by different researchers [Long and Neale, 1993; Preece et al., 1993a; Lunardhi and Passino, 1991; Plant, 1992; Vinze, 1992]. We define performance of a rule-based system in terms of two measures: the ability of the system to arrive at a solution for a given problem, and the utilization value of its resources used in problem solving. These measures are indicative of how adequate is a system in its domain [Giovanni, 1989], and how efficiently it solves problems in its domain; they are called **adequacy** and **optimality** respectively. In particular, the adequacy of the system indicates the extent a user can rely on the system to solve a given problem. Though optimality and adequacy are general measures of performance, they are relative to the goal specification of the domain.[3] Note, the use of paths and goals in assessing system qualities (section 4.6) can also be viewed as qualitative measures of performance when comparing two systems.

General measures to quantify performance are needed because generic conclusions about system performance cannot be made using specific coverage data obtained from testing alone (even after testing with a large number of test cases). Though conventional testing and test coverage measures produced as part of testing a system can provide data to analyze the run-time performance, the data is relative to a set of test cases. In addition, this set of test cases is not guaranteed to be a representative set of the domain (one that ensures that the system has been tested sufficiently) [Preece and Shinghal, 1992; Chang et al., 1990; O'Keefe et al., 1987].

[3]This is not a drawback because the goal specification of a domain is meant to abstract general problem solving for the domain and not that of solving one specific problem.

Test coverage with one test case. (File: log.a.patra)

Strategy	0%	1-50%	51-60%	61-70%	71-80%	81-90%	91-99%	100%
Conservative	72.5	17.6	0.8	0.8	0.2	0.0	0.0	8.1
Moderate	67.4	8.5	2.3	6.6	6.0	1.0	0.0	8.1
Liberal	67.4	7.4	0.0	1.2	3.9	3.1	0.0	17.1

Test coverage with seventeen test cases. (File: log.q.patra)

Strategy	0%	1-50%	51-60%	61-70%	71-80%	81-90%	91-99%	100%
Conservative	35.9	31.8	6.2	6.2	1.4	1.0	0.2	17.4
Moderate	28.3	21.1	3.7	10.5	8.9	8.5	0.6	18.4
Liberal	28.3	18.8	2.7	5.0	2.9	7.2	1.7	33.3

Figure 4.17: *Percentages of Blackbox expert paths covered in testing: two samples.*

For example, consider Figure 4.17 that depicts the coverage information obtained for the Blackbox expert (section 4.3) for one test case and seventeen test cases [Preece et. al., 1993b]. Using a conservative counting strategy, the percentage of paths that were covered 100% for the Blackbox expert was 8% for a single test case, and goes up to only 17% with seventeen test cases. Thus, even with 17 test cases, nearly two-thirds of the paths are not covered.[4] However, we cannot assume that the system does not exercise a major portion of its paths or a large portion of the rule base is redundant, even though this set of test cases was used to compare its performance with human experts [Grossner et al., 1991], because this is not necessarily a representative set of test cases that are needed to sufficiently test the system [Preece et al., 1993b, 1995].

General performance measures are necessary when comparing two systems because of the following:

- data observations based on a set of test cases cannot compare systems operating in different domains; and
- testing alone can miss detecting some general performance attributes of a system [Chander et al., 1994].

To model the optimality and adequacy of a system, consider a typical problem solving scenario in this framework. Let γ be a set $\{i_1, i_2, \ldots, f_1, f_2, \ldots\}$ such that in

[4]At the other extreme, the number of paths that had 0% coverage dropped from 70% (one test case) to 35% (seventeen test cases), but, based upon this information, one cannot conclude that nearly one-third of the rule base is useless.

solving a problem P in the domain, the system infers goals $i_1, i_2, \ldots, f_1, f_2, \ldots$ where,

1. The f's are final goals that constitute the problem solution.

2 The i's are intermediate goals that are ancestors of the final goals inferred.

In solving a problem, if an intermediate goal i' was inferred that later did not become an ancestor of any of the f's, then i' does not belong to γ. Since i' did not help in inferring any of the f's, inferring i' was unnecessary. Thus, the set of i's that is not an ancestor to any of the final goals indicate that the system is not functioning optimally in its domain. Similarly, the problems for which the system does not infer a solution indicate that the system is not adequate enough for the domain.

More precisely, the performance measures adequacy and optimality can be computed using the notions of **goal relevancy** and **relevancy ratio**. Given the initial evidence corresponding to a problem, if the system infers some final goals that correspond to the solution of the problem, then the inferred final goals are said to be relevant. An intermediate goal is relevant iff there exists a problem in the domain whose solution has this intermediate goal as its ancestor. A final (intermediate) goal is irrelevant iff it is not a relevant final (intermediate) goal. A system is **goal-relevant** iff all its goals are relevant. A system is said to be **goal-irrelevant** iff it is not goal-relevant. Clearly, an irrelevant final goal does not constitute a solution to any problem in the domain and reflects on the capability of the system, that is, its possible inadequacy [Giovanni, 1989]. Irrelevant intermediate goals indicate that the system at times may do redundant work by inferring goals that are never used for inferring any solution; this may suggest that the system may be functioning sub-optimally in its domain. The relevancy ratio measures the percentage of relevant intermediate and final goals, and indicates the extent of goal (hence, resource) utilization of the system.

Definition 16 (Relevancy ratio) *The relevancy ratio of intermediate goals is defined as* $\frac{NI_r}{NI_t}$ *where,* NI_r *is the number of relevant intermediate goals and* NI_t *is the total number of intermediate goals in a goal specification. The relevancy ratio of final goals is defined as* $\frac{NF_r}{NF_t}$ *where,* NF_r *is the number of relevant final goals and* NF_t *is the total number of final goals in a goal specification. The overall relevancy ratio of the*

i_1
i_2
i_3
f_1
f_2
i_4
i_5
f_3
$\vdots$

Goals observed in testing the system on problem P_j.

The set γ (partial) from this test is,

$$\gamma_j = \{ \langle\{i_1, i_2, i_3\}, f_1\rangle, \langle\{i_1, i_2, i_3\}, f_2\rangle, \langle\{i_1, i_2, i_3, i_4, i_5\}, f_3\rangle \}$$

This set γ_j extracted is partial because, this set of goals inferred is specific to problem P_j. We have used the notation $i_1, i_2, \ldots$ to indicate intermediate goals and $f_1, f_2, \ldots$ to indicate final goals.

Figure 4.18: *Determination of optimality and adequacy by testing with a set of test cases.*

system is defined to be:

$$\Re = \frac{(NI_r + NF_r)}{N}$$

where N is the total number of goals in the goal specification (that is, $N = NI_t + NF_t$).

A relevancy ratio of less than 1 for intermediate (final) goal indicates sub-optimality (inadequacy). Thus, the relevancy ratio $\Re$ for the system is a general performance measure of the system's utilization of its resources for problem solving. Note, if the adhocness metric of a scheme is ∞ (chapter 3, section 3.3.2), rule bases adhering to that scheme may exhibit a low relevancy ratio.

One way of determining goal relevancy and relevancy ratio of a system is by testing the system on a set of problems $P_1, P_2, \ldots$ to determine the set γ for each problem P_j as shown in Figure 4.18, and computing their union. However, testing to determine a system's optimality and adequacy is limited by the following result.

Theorem 2 *The determination of goal-relevancy by testing (that is, using a set of test cases) is semi-decidable.*

Proof. *For every P_i used as a test case, we obtain the corresponding set γ_i by the procedure outlined above. If a stage comes that $\gamma_1 \cup \gamma_2 \cup \gamma_3 \cup \ldots$ contain all the goals in the system, then we terminate the procedure since the system is goal-relevant. The disadvantage of this approach is that this procedure may* **never** *terminate. The reason* *is as follows: if after generating γ_i, the procedure has not terminated because the union of the γ's collected so far does not include all the goals, this does not necessarily suggest that this procedure will terminate after generating γ_{i+1}. The rapidity with which the procedure terminates, if it does at all, depends upon the problems selected from the domain which can be infinite. The procedure is thus semi-decidable in general.*

Note that the semi-decidability arises owing to the decision of using a testing strategy that tries to include all problems in the domain to determine the performance measures optimality and adequacy; the problem of optimality and adequacy determination itself is not semi-decidable. However, most developers tend to use a functional testing approach for their systems and often refrain from performing any structure-based testing as observed in [Hamilton et al., 1991] (see also chapter 1, section 1.2.2). Thus, the above result is important to a developer because it emphasizes the need to determine the performance measures optimality and adequacy through an examination of the system structure by pointing out the limitation of the commonly used functional testing in this context.

The determination of relevant and irrelevant goals is non trivial in general because a given intermediate goal may have a path inferring the goal, yet it can be irrelevant. This can happen because this goal was never used to infer any other final goal. Similarly, a final goal may be inferred by a path, but the rules in the path may never be enabled because no permissible combination of initial evidence would be causal to do so. Thus, it is necessary to enumerate all path sequences from permissible combinations of initial evidence to final goals to determine the goal dependencies. In other words, a procedure to extract the goal graph from a rule base, using paths as connectors, is required. The goal graph extracted from the rule base can be traversed to identify goal irrelevancy in the system, if any.

A goal graph traversal should mark all the connector chains from permissible initial evidence to a final goal. Since these are simply the routes (definition 15, page 84) paths and goals can be used to determine the goal-relevancy of a system. Intuitively, the procedure to determine the goal-relevancy of a system proceeds as shown in Figure 4.19.

1. Select a final goal f.
2. Starting from f, recursively traverse goal graph containing all the goals from which f can be inferred.

The above procedure gives all relevant routes, and irrelevant routes culminating on a final goal in the goal graph.
To obtain irrelevant routes that culminate on an intermediate goal, do the following:

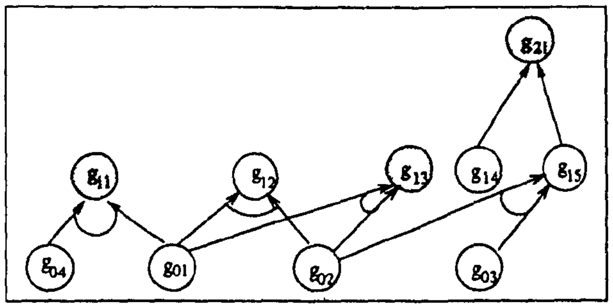
1. Start from every intermediate goal i that has not appeared in any of the routes after steps 1 and 2 of the above procedure are applied to all the final goals.
2. For such intermediate goals i traverse recursively the goal graph containing all the goals from which i can be inferred.

This would obtain irrelevant routes that culminate on the intermediate goal i.

Figure 4.19: *Determination of relevant and irrelevant routes in a goal graph.*

The goal graph of the example rule base shown in Figure 4.7 is shown in Figure 4.20. For this rule base and the given goal specification, the algorithm would output intermediate goals g_{12} and g_{14} as irrelevant. An examination of the rules realizing these goals indicate a design scheme violation (goal g_{12} does not contain any intermediate hypothesis, thus violating restriction I3), and errors (goal g_{14} is irrelevant because rule R15 can never be enabled from the given set of initial evidence possibly indicating a deficient goal specification). Note, had the rule base been coded such that rules R16 and R17 were combined into one rule, then solution g_{21} would have been flagged as irrelevant. The relevancy ratio of the example system is 80%: this means that eighty percent of the goals specified were usefully utilized for problem solving.

For the Blackbox expert, 38.7% of the goals specified were not inferred by any path. This is not surprising as the goals were reverse engineered from the rule base and goal specification refinement was focused only to control the computation; thus, many of the goals specified were not useful in abstracting problem solving in its domain.



$$
\begin{aligned}
g_{01} &= \text{REGISTERED}(x) \\
g_{02} &= \text{GREENBORDRID}(x) \\
g_{03} &= \text{HARDWORKING}(x) \\
g_{04} &= \text{DEANSLIST}(x) \\
g_{11}^{*} &= \text{GOODGRADES}(x, y) \wedge \\
&\quad\ \text{GT}(x, Gpa, 3.5) \\
g_{12} &= \text{ACADEMIC}(x) \\
g_{13}^{*} &= \text{ACADEMIC}(x) \wedge \text{YOUNG}(x) \\
g_{14} &= \text{BURSARY}(x) \\
g_{15} &= \text{GOODCAREER}(x) \\
g_{21}^{*} &= \text{COMFLIFE}(x)
\end{aligned}
$$

Figure 4.20: *The goal graph for the rule base shown in Figure 4.7.*

This further emphasizes that system development should start with an initial goal specification, and as part of the system's life-cycle, incremental evolution of the rule base and refinement of goal specification should go hand in hand to improve its quality and performance.

A formal presentation of the algorithm to traverse a goal graph using the connectors is shown in Figure 4.21. Once the irrelevant goals are identified, the relevancy ratio can be determined. In the worst case, the algorithm visits every intermediate goal $|F|$ times, where $|F|$ is the number of final goals, because every final goal $f \in F$ can require every intermediate goal in I. Thus, the total number of times the intermediate goals would be visited in the worst case by the traversal algorithm is given by $|I| * |F|$. In this analysis, we have not included the complexity of goal graph extraction which can be exponential in the worst case as outlined by the following result.

Theorem 3 *Given an arbitrary rule base and goal specification, extraction of the goal graph from the rule base is of exponential complexity in the worst case.*

Proof. *The proof consists in showing that the procedure for extracting the necessary paths to build the goal graph is identical to a procedure that has an exponential time complexity in the worst case for at least one instance of the goal specification. Since the*

```
Procedure Get$\gamma$
Input: Connectors, Set of final goals ($F$), Set of intermediate goals ($I$)
Output: The set $\gamma$
begin /* Determination of $\gamma$ */
    1. $\gamma := \emptyset$

    2. Start by selecting a $f \in F$; mark connectors to $f$ as unseen

    3. Repeat
            3.1 While (one more connector $\Phi$ to $f$ is marked unseen) do
                    3.1.1. $I' := \emptyset$

                    3.1.2. Pick the connector $\Phi$ inferring $f$ marked unseen;
                    mark this connector as seen.

                    /* obtain the goals at the other end of $\Phi$ */
                    3.1.3. Derive the subset $G$ of $I$ that is required by
                    the connector $\Phi$

                    3.1.4. $I' := I' \cup G$ /* $G$ obtained in previous step */

                    /* Traverse the goal graph using the connectors */
                    3.1.5. Apply 3.1.3—3.1.4 recursively for all connectors to each
                    goal in $I'$ until a level-0 goal is reached.

                    /* Accumulate the goals collected so far into $\gamma$ */
                    3.1.6. $\gamma := \gamma \cup I' \cup \{f\}$
            end /* while */
            /* All possible traversals from $f$ has been completed */

    Until (as long as there is a goal in $F$ to which step 3.1 has not been applied)

    4. Output $\gamma$.
end. /* End of Get$\gamma$ */
```

Figure 4.21: *A procedure to determine whether a system is goal relevant by traversing the extracted goal graph from a rule base.*

goal specification can be arbitrary, in a goal specification where every final hypothesis is a final goal and every intermediate hypothesis is an intermediate goal, the goal graph extraction procedure is identical to computing final-hypothesis labels from initial evidence combinations [Ginsberg, 1988; Loiseau and Rousset, 1993] which has an exponential complexity in the worst case.

Such a worst case scenario, however, would rarely arise in practice when using a well formulated goal specification. Theorem 3 is only a caution against ad hoc goal specification, and should not be construed to be a limitation of this framework.

4.5 Verification of Rule-based Systems

In practice, a system should be verified before it is validated because the anomalies in its rule base (if any) can affect the validation results [Preece, 1992]. Rule-based systems inadvertently suffer from four anomalies known as CARD (chapter 1, section 1.2.2): Circularity, **A**mbivalence, **R**edundancy, and **D**eficiency. These anomalies can cause errors during rule base processing. It is not practical to manually detect these anomalies in large rule bases; procedures to automate this detection are required. Such procedures are collectively referred to as rule base verification procedures [Preece et al., 1992b].

A rule base that contains a rule and/or an atom such that removing it, or removing a part of it does not affect the functioning of the system, is said to exhibit redundancy. A system is said to be ambivalent whenever an inviolable becomes true. Deficiency refers to missing knowledge in a system. Circularity is present whenever circular dependencies exist between the rules [Preece et al., 1992a]. In order to detect these anomalies, one must identify certain rule situations that can be indicative of one or more of these anomalies. They are called **rule aberrations** because they indicate an abnormality in the system [Chander et al., 1995].

Definition 17 (Rule Aberration) *Let $\mathcal{A}$ denote the anomaly set of a rule base (typically the CARD anomalies). A rule aberration in a rule base consists of a set of paths that portray the manifestation of one or more elements from the anomaly set $\mathcal{A}$. If*

π, a (sub)set of paths in a rule base, is an aberration, then we can state that

$$\pi \models a \subseteq \mathcal{A}$$

Goal specification, paths, and the extracted goal graph allow for a comprehensive detection of the CARD anomalies in a rule base at any stage during its construction: simply spot paths satisfying the conditions of one, or more rule aberrations. We give below a list of aberrations, and provide comments about the possible anomalies that the aberration could indicate. We, however, do not claim that the list is exhaustive. Studying the aberrations provides a different perspective on anomaly detection in rule bases by basing this process on rule sequences pertinent to problem solving rather than rules, thus capturing the rule interactions as well.

Prior to checking for the aberration conditions in a rule base, it is assumed that the relevant and irrelevant routes have been extracted from the ıle base (section 4.4). The identification of specific rules and atoms that cause anomalies in the rule base is called **flagging**. These procedures flag rules to make the knowledge engineer aware of them; on further examination, the knowledge engineer may leave a flagged rule unchanged, edit the flagged rule, or may add other rules to the rule base so that the flagged rule is no longer causal to the detected anomalies. Below, we describe a list of aberrations that characterize redundancy [Chander et al., 1995]. Note, these aberrations can be caused due to deficiency in the system as well.

Aberration 1. If a rule appears only in irrelevant routes, then it does not contribute to solving any problem because either it does not begin from initial evidence, or it does not end in a solution. Hence, such a rule could be redundant. Similarly, a rule does not contribute to problem solving if it does not appear in any route. Such rules are flagged by these procedures. The description for this aberration is shown in Figure 4.22.

In the example rule base, rules R1, R2, and R3 (see Figure 4.8) will be flagged because they appear only in an irrelevant route, and rule R15 will be flagged since it appears in no route (the atom `GOODRECORD(x, Juniorcollege)` in its antecedent is not inferred by any of the rules). Interestingly, even if this is corrected, rule R15 would

Aberration 1 (Redundancy due to route irrelevancy.)

```
begin
 1. For all rules $r$ do
        if all routes in which $r$ appears are irrelevant or empty,
        then flag $r$ as potentially redundant;
end
```

Figure 4.22: *A description of rule aberration 1.*

Aberration 2. (Detection of redundant rule chains.)

```
begin /* For redundancy of rule chains; See Nguyen (1987);
  we also enable detection of redundant rules */
  1. Flag paths appearing only in irrelevant routes as redundant.
  2. For any two paths $\Phi_1$ and $\Phi_2$,
    (i) if the goals required for $\Phi_1$ subsume the goals required for $\Phi_2$, and
    the goal inferred by $\Phi_1$ subsumes the goal inferred by $\Phi_2$, then flag $\Phi_1$ and $\Phi_2$.
/* For efficiency, we may consider only relevant routes */
  3. For any two paths $\Phi_1$ and $\Phi_2$ where each appears in at least one relevant route
    (i) if goals required for $\Phi_1$ is subsumed by the goal(s) required for $\Phi_2$, and
    (ii) if goal inferred by $\Phi_2$ subsumes goal inferred by $\Phi_1$, then flag $\Phi_1$ and $\Phi_2$;
/* For example, if goal atoms inferred by $\Phi_2$ is a subset of the goal atoms
inferred by $\Phi_1$, then flag rules in $\Phi_2$ not appearing in any other path. */
end
```

Figure 4.23: *A description of rule aberration 2.*

still not appear in any path. Inspection should reveal that the atom GOODCAREER(x) in the consequent of R15 is redundant.

Aberration 2. A rule chain as used in the literature [Nguyen, 1987] is a linear sequence of rules. Aberration 2, shown in Figure 4.23, can be used to detect redundant rule chains.

We detect redundant rule chains without much of computational overhead. For example, we can flag the sequence $a \rightarrow c \rightarrow d$ as redundant with respect to $a \rightarrow b \rightarrow d$. Further, if rule $c \rightarrow d$ does not appear in any path other than those where $a \rightarrow c$

Aberration 3. (Detection of redundant atoms.)

```
begin /* Based on goal redundancy */
  1. For all paths $\Phi$, if more than one rule infers the set of goal atoms for goal $g$
  inferred by $\Phi$ do
    (i) Let $X$ := set of goal atoms that are multiply inferred.
    (ii) if any subset Y of $X$ is multiply inferred in all paths in which these goal atoms,
      are inferred, flag this subset of goal atoms.
    (iii) if a toe rule $r$ in $\Phi$ has goal atoms only from the set $Y$ in (ii) above,
      flag rule $r$.
  /* a complementary step to step 1; applied to non-goal atoms; */
  2. For all paths $\Phi$, let $X$ be the set of dangling non-goal atoms
    (i) if non-goal atoms in (a subset of) $X$ are dangling in every path they appear,
      flag this (sub)set of non-goal atoms.
    (ii) For all rules $r_1$ that infer some dangling non-goal atoms in (a subset of) $X$
    above, and some other consumed non-goal atoms, if some other rule(s) infer the
    consumed non-goal atoms in every path where $r_1$ appears, flag $r_1$.
end
```

Figure 4.24: *A description of rule aberration 3.*

appears, we can infer that the former rule is redundant; in addition, if rule $c \rightarrow d$ appears in only those paths where $a \rightarrow c$ appears, we can also conclude that $a \rightarrow c$ is possibly redundant. The method in Nguyen (1987) will flag the above rule chains, but their approach to the extraction of rule chains may not be practical for large rule bases. We, however, can extract paths from large rule bases using the **path hunter** tool (section 4.3.1). Note, by virtue of path definition, we flag not only rule chains that are linear, but those that are non-linear as well. In the example rule base, path $\Phi 1$ (see Figure 4.8) exhibits redundancy with respect to path $\Phi 2$: both $\Phi 1$ and $\Phi 2$ infer `ACADEMIC(x)` whenever initial evidence { `REGISTERED(x)`, `GREENBORDRID(x)` } is present; path $\Phi 1$ is a non-linear sequence of rules.

Aberration 3. The intuition behind aberration 3, shown in Figure 4.24, for detecting redundant atoms is that whenever atoms are inferred by a rule r, if some other rule(s) always fire to infer these atoms additionally, then such atoms in the consequent of r are possibly redundant.

Two types of non-goal atoms can be identified in a path based upon the consumption of a rule in that path: "dangling" atoms and "consumed" atoms. A dangling non-goal atom in a path is an atom in the consequent of a rule, but does not unify with an atom in the antecedent of any other rule in the path: for example, atom `COMPLETED(x, Juniorcollege)` in path Φ3 of Figure 4.8. A consumed non-goal atom in a path is an atom that is in the consequent of a rule, and unifies with an atom in the antecedent of one other rule in the path. Toe rules can be easily flagged redundant by this aberration whenever their consequent is inferred by other rules in every path they appear. All that is required is a simple look up on the set of paths. In the example rule base, the rule R4 will be flagged redundant by step 1 (iii) of Figure 4.24 because the atom `ACADEMIC(x)` in its consequent is always inferred by some other rule in every path in which it appears, and atom `COMPLETED(x, Juniorcollege)` in the consequent of rule R9 to be redundant by step 2 (i) of Figure 4.24. Note, atom `COMPLETED(x, Juniorcollege)` is used in the antecedent of rule R15; thus, detection methods based on simple unification may not detect this atom as redundant [Polat and Guvenir, 1993].

Aberration 4. The traditional methods flag duplicate rules and rules of the form $a \rightarrow b$ and $a \wedge c \rightarrow b$, where the latter is subsumed by the former. Using paths, however, a more general form of detection is possible. Let $r_1 : A_1 \wedge A_3 \rightarrow H$ and $r_2 : A_1 \wedge A_2 \rightarrow H'$ be any two rules such that A_3 is a goal atom, the remaining atoms are non-goal, and the H's represent hypotheses which can be a conjunction of atoms. Further, let H subsume H'. Then, the rule pair $< r_1, r_2 >$ is flagged redundant based upon the following conditions:

(i) If every path in which r_1 appears has at least one rule that infers A_2 (in other words, whenever r_1 fires, r_2 can also fire) then perhaps r_1, or r_2 is redundant.

(ii) Whenever r_2 occurs in a path from G to g and A_3 is contained in G (in other words, r_1 can also be used to traverse between any G to g whenever r_2 can do so), then perhaps r_1, or r_2 is redundant.

Aberration 4 (Subsumed rules.)

```
begin
  Let $r_1$ and $r_2$ be any two rules in the rule base.
  1. Flag $< r_1, r_2 >$,
  /* $r_1$ can traverse between any goal-goal whenever $r_2$ can */
    (i) if the goal atoms in the antecedent of $r_1$ are contained in the
    goals required for every path where $r_2$ appears, and
    (ii) Every non-goal atom in the antecedent of $r_1$ not present in the
    antecedent of $r_2$ is supplied by some rule in every path in which $r_2$ appears, and
    (iii) One of the rule consequents subsumes another.
  2. Flag $< r_1, r_2 >$, /* vice versa */
    (i) if goal atoms in the antecedent of $r_2$ are contained in the set of
    goals required for every path that contains $r_1$, and
    (ii) Every non-goal atom in the antecedent of $r_2$ not present in the
    antecedent of $r_1$ is supplied by some rule in every path in which $r_1$ appears, and
    (iii) One of the rule consequents subsumes another.
end
```

Figure 4.25: *A description of rule aberration 4.*

We flag both the rules because the consequent of rules r_1 and r_2 should be examined before concluding redundancy. For example, (1) if H is $BIRD(x)$ and H' is $BIRD(Tweety)$, and condition (i) above holds then r_2 is redundant; and (2) if H is $BIRD(x)$ and H' is $BIRD(x) \wedge SINGS(x)$, and condition (ii) above holds then r_1 is redundant. A description of this aberration is shown in Figure 4.25.

As a further example, the above aberration can be used to flag rules R8 and R13 in the rule base of Figure 4.7 because rule R13 is enabled whenever rule R8 is enabled, and the consequent of R13 contains the consequent of R8. This unwanted interaction (due to subsumption) between these rules accounts for the coverage of path $\Phi4$ for test set 2 shown in Figure 4.9 (section 4.3). The detection of subsumed rules is important because rule subsumption can interfere with certain "greedy" inference strategies producing unexpected results [O'Leary, 1995].

Aberration 5. Inferring an atom used only in the antecedent of flagged rules is possibly redundant. If an atom A in the antecedent of one of the rules flagged by

Aberration 5. (Detecting useless inferences.)

```
begin
  1. Let $X$ := rules flagged redundant in any of the above aberrations 1–4, consider
  the atoms in the antecedent of the rules in $X$.
  2. If some of these atoms never appear in the antecedent of rules not in $X$
  then flag these atoms.
  3. If a rule $r$ infers any of the atoms flagged in step 2,
  then flag $r$. /* useless inference */
end
```

Aberration 6. (Detecting redundant consumed atoms.)

```
begin
  1. Let $X$ := rules flagged redundant in any of the above aberrations 1–4, consider
  the atoms in the consequent of the rules in $X$.
  2. If some of these atoms never appear in the consequent of rules not in $X$,
  then flag these atoms.
  3. If a rule $r$ uses any of the atoms flagged in step 2,
  then flag $r$. /* potential unreachability */
end
```

Figure 4.26: *A description of rule aberrations 5 and 6.*

aberrations 1–4 never appears in the antecedent of rules that are not flagged, then this atom is redundant. Aberration 5 checks for this condition, and is described in Figure 4.26. Note, this type of flagging would require examination of some rules that have not been flagged, but use these atoms in their consequent.

In the example rule base, atom HONSCOURSES(x) would be flagged by aberration 5. This would, in turn, require examination of rules R7 and R11 that use this atom in their consequent.

Aberration 6. Atoms in the antecedent of a rule r inferred only by rules flagged already are redundant. The rule r can become unreachable if all the flagged rules are deleted from the rule base. Aberration 6 checks for this condition and is shown in Figure 4.26. Note, this type of flagging would require examination of some rules that have not been flagged, but use these atoms in their antecedent. This is complementary

Flagged Rules/Atoms	Route(s)	Aberration(s)
R1, R2, R3 R17 R15	$\Phi1$ $\Phi6$ none	Aberration 1
Paths $\Phi1$ and $\Phi2$	NA	Aberration 2
COMPLETED(x, Juniorcollege) R4, ACADEMIC(x)	NA	Aberration 3
R8, R13	NA	Aberration 4
UGRAD(x), HONSCOURSES(x),...	NA	Aberrations 5 and 6

Figure 4.27: *Summarizing the results of redundancy/deficiency aberrations for the example rule base of Figure 4.7.*

to aberration 5. Checking for aberrations 5 and 6 is, however, optional.

In the example rule base, atom UGRAD(x) would be flagged by aberration 6. This in turn requires examination of rule R9 that uses this atom in its antecedent. For convenience, the rule aberrations and the affected rules and atoms in the example rule base are summarized in Figure 4.27.

Inspection of the rules and atoms flagged by the above aberrations should also reveal a design scheme violation. Of course, whenever rules or atoms are flagged by aberration procedures, refining the goal specification and editing the rule base may remove the anomalies. For example, the subsumption problem between R8 and R13 can be corrected by modifying the permissible combination of initial evidence HARDWORKING(x) $\wedge$ GREENBORDRID(x) to HARDWORKING(x) $\wedge$ GREENBORDRID(x) $\wedge$ GOODRECORD(x,y), and the antecedent of rule R13 to contain the atom GOODRECORD(x, Courses). However, modifications to fix a detected anomaly can also introduce additional anomalies. For illustration, consider the following modification to the rule base in Figure 4.7 to ensure that rule R15 appears in a path.

As rule R15 was flagged redundant by aberration R-1, a knowledge engineer trying to fix this redundancy can include GOODRECORD(x, Juniorcollege) as initial evidence. However, this still would not make R15 appear in a path. It is required to remove GOODCAREER(x) in the consequent of rule R15. In addition, the conjunction

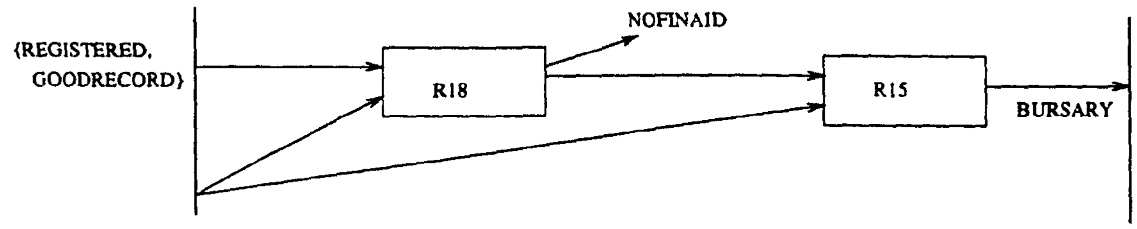


Figure 4.28: *The path where rule R15 appears.*

`REGISTERED(x)` ∧ `GOODRECORD(x,y)` must also be added as a permissible combination of initial evidence. To accommodate the status of a registered person with good grades in junior college, assume that the following new rule is added:

$$\text{R18: REGISTERED}(x) \wedge \text{GOODRECORD}(x, \textit{Juniorcollege}) \rightarrow$$
$$\text{COMPLETED}(x, \textit{Juniorcollege}) \wedge \text{NOFINAID}(x)$$

to encode the knowledge describing a person just registered after completing junior college, who does not receive any financial aid. This adds a new path which is shown in Figure 4.28.

However, the modification to remove the redundancy of rule R15 as now resulted in another anomaly in the rule base. The path in Figure 4.28 infers an inviolable: `BURSARY` ∧ `NOFINAID`; thus, the system is now ambivalent. An inspection of the rule base shows that, in this case, atom `NOFINAID(x)` is not required in the consequent of rule R18, because rule R1 makes that inference for a general registered undergraduate student. The impact of a rule modification can immediately be assessed by thus checking the paths affected for aberrations, if any.

In general, ambivalence can manifest in other ways than the one described above. The aberrations shown in Figure 4.29 characterize ambivalence in a system [Chander et al., 1995b, 1995c].

Aberration 7. This is based on a path and a goal it infers. Clearly, in trying to infer a goal, the non-goal atoms in a path should not violate a constraint. For example, `MALE(x)` ∧ `FEMALE(x)`, should not be inferred in a path even though it may infer something useful. Condition 2 of this aberration prohibits incorrect goal

Aberration 7 (Ambivalence in a Path.)

```
begin
  1. (Intra-path) The conjunction of non-goal atoms in a path should not be
  subsumed by an inviolable. This also applies to the conjunction of all the atoms
  in a path.
  2. (Goal ambivalence) No goal should be subsumed by an inviolable. This can
  indicate inaccuracies in knowledge acquisition.
  3. The set of goals required by a path should not contain or be subsumed by
  an inviolable.
end.
```

Aberration 8 (Ambivalence over Inference Chains.)

```
begin
  1. The conjunction of non-goal atoms in a relevant route should not be subsumed by
  an inviolable. This means as part of problem solving at least one constraint
  is violated.
  2. The conjunction of goal atoms in a relevant route should not be subsumed
  by an inviolable. This means as part of problem solving at least one constraint is
  violated.
  /* Also check for constraint violation for all atoms inferred in the route */
end.
```

Aberration 9 (Impact of Impermissible Initial Evidence.)

```
begin
  1. As part of route determination check if an impermissible set of initial evidence
  is obtained at level-0.
  2. (Reverse of 1) For every set of impermissible initial evidence, check if paths
  and routes can be enumerated. Flag all these paths.
end.
```

Figure 4.29: *A description of rule aberrations 7, 8, and 9.*

specification, and condition 3 checks that no path should start at the cost of violating a constraint.

Aberration 8. Aberration 8 considers transitivity of inferences—an inference used for other inferences—to check for ambivalence. Note, checking that every path is free from ambivalence (cf. aberration 7) does not ensure that a route is free from ambivalence. There are two scenarios to consider when checking for route ambivalence. In the first scenario, while goals are not inviolables themselves, some of the goal atoms can be part of inviolables. During problem solving, whenever this set of atoms are collectively inferred over a sequence of paths, the system would infer an inviolable (hence, ambivalent). In general, a system is potentially ambivalent, whenever a set of goals subsume an inviolable. The ambivalence is potential because this is problematic iff we have path(s)/routes that involve this set of goals. In the second scenario, we ensure no atoms involved in a relevant route violate a constraint: this ensures that this complete sequence of paths from level-0 goals to final goals is free from ambivalence. More specifically, we may check if a set of non-goal atoms, or goal atoms (or their combination) violates a constraint in order to focus the fix in the rule base, or goal specification (or both).

Aberration 9. This is based upon measuring the system response to impermissible combinations of initial evidence by checking the rules/paths affected. Conditions 1 & 2 of this aberration check for all routes that can be caused by an impermissible combination of initial evidence: this is serious because if at least one of these routes is relevant, then initial evidence that is insufficient (or, perhaps invalid) is treated as a valid input by the system. This reflects on inaccuracy and negative adequacy of the system (solving problems that are not intended to be solved). For instance, a doctor requires more information in addition to the fact that *x is female* to infer that *x is pregnant*. However, if a system infers `PREGNANT(x)` whenever `FEMALE(x)` is input, this indicates an error in knowledge encoding. The purpose of aberration 9 is to detect such behavior, if present, in the system. As an additional example, the rule base of

Aberration 10 (Path Circularity.)

```
begin
  1. A sequence of paths ($\Phi_1 \ldots \Phi_n$) where goal inferred by a path $\Phi_{i-1}$ $1 < i \leq n$ is
  used as part of the start state of path $\Phi_i$, such that the goal inferred by $\Phi_n$ is
  contained in the start state of $\Phi_1$.
end.
```

Figure 4.30: *A description of rule aberration 10.*

Figure 4.7 can infer a solution COMFLIFE(x) from DEANSLIST(x).[5] However, as atom DEANSLIST(x) alone does not constitute a permissible combination of initial evidence according to the goal specification, this can indicate an error in the rule base and/or goal specification. In our case, it indicated an error in the rule base (deficiency in rule R13) and goal specification. Note, checking for condition 2 of aberration 9 only requires a minor modification to the algorithm described in Figure 4.21 (section 4.4).

Aberration 10. There is only one aberration to characterize circularity in a rule base. Its description appears in Figure 4.30. The start state of a path refers to the set of goals required by a path before the rules in the path can fire. The above detection of circularity can flag more than one circular dependency between the rules in the system because a path is not necessarily a linear sequence of rules. (Any cycle detection algorithm for graphs can be used to detect cycles once the goal graph is extracted from the rule base.) In addition, during path extraction, path hunter (section 4.3.1) also flags rules whenever it detects a circular accessibility relationship between the rules in the rule base. The algorithms for detecting the rule aberrations described in this section appears in the appendix. The final version of the rule base after making the required modifications to fix the detected anomalies is shown in Figure 4.31.

Once all the detected anomalies are fixed, and the system behavior is judged to be acceptable by a knowledge engineer, the system is tested for the satisfaction of the **user acceptability criteria** [Preece, 1990; Ghezzi et al., 1991]. Often, the user

[5]This rule chain was captured as a fragment by path hunter.

$R1$:	$\mathrm{REGISTERED}(x) \wedge \mathrm{GREENBORDRID}(x) \rightarrow \mathrm{ENROLLED}(x) \wedge \mathrm{NOFINAID}(x)$
$R2$:	$\mathrm{ENROLLED}(x) \rightarrow \mathrm{STUDENT}(x)$
$R3$:	$\mathrm{GREENBORDRID}(x) \wedge \mathrm{NOFINAID}(x) \rightarrow \mathrm{UGRAD}(x)$
$R4$:	$\mathrm{STUDENT}(x) \wedge \mathrm{UGRAD}(x) \rightarrow \mathrm{ACADEMIC}(x) \wedge \mathrm{YOUNG}(x)$
$R5$:	**DELETED**
$R6$:	**DELETED**
$R7$:	$\mathrm{DEANSLIST}(x) \rightarrow \mathrm{HIGHGPA}(x) \wedge \mathrm{HONSCOURSES}(x)$
$R8$:	$\mathrm{REGISTERED}(x) \wedge \mathrm{HONSCOURSES}(x) \rightarrow \mathrm{UGRAD}(x)$
$R9$:	$\mathrm{UGRAD}(x) \wedge \mathrm{DEANSLIST}(x) \rightarrow \mathrm{GOODGRADES}(x, Juniorcollege)$
$R10$:	$\mathrm{HIGHGPA}(x) \wedge \mathrm{REGISTERED}(x) \rightarrow \mathrm{GT}(x, Gpa, 3.5)$
$R11$:	$\mathrm{HARDWORKING}(x) \wedge \mathrm{GREENBORDRID}(x) \rightarrow \mathrm{HIGHGPA}(x) \wedge \mathrm{HONSCOURSES}(x)$
$R12$:	**Merged with rule R11 above**
$R13$:	$\mathrm{HIGHGPA}(x) \wedge \mathrm{HONSCOURSES}(x) \wedge \mathrm{GOODRECORD}(x, Courses) \rightarrow \mathrm{DISTINCTION}(x)$
$R14$:	$\mathrm{DISTINCTION}(x) \rightarrow \mathrm{GOODCAREER}(x, Industry)$
$R15$:	$\mathrm{GOODRECORD}(x, Juniorcollege) \wedge \mathrm{COMPLETED}(x, Juniorcollege) \rightarrow \mathrm{BURSARY}(x)$
$R16$:	$\mathrm{GOODCAREER}(x, Industry) \rightarrow \mathrm{COMFLIFE}(x)$
$R17$:	$\mathrm{BURSARY}(x) \rightarrow \mathrm{COMFLIFE}(x)$
$R18$:	$\mathrm{GOODRECORD}(x, Juniorcollege) \wedge \mathrm{REGISTERED}(x) \rightarrow \mathrm{COMPLETED}(x, Juniorcollege)$

Figure 4.31: *The modified rule base based upon the evaluation results of the rule base shown in Figure 4.7.*

acceptability criteria can impose further constraints on the user interface required, on the response times, on the system adequacy, etc. Let us assume that the user acceptability criteria involves that the system should be adequate in its domain (that is, should produce a solution for every permissible combination of initial evidence).

The final set of paths and the goal graph for the system are shown in figures 4.32 and 4.33. There are no more unwanted rules or atoms. All the rules in the rule base come into play for problem solving and the system is optimal and adequate for the given goal specification. As there are no irrelevant final goals, the system satisfies the user acceptability criteria.

4.6 Using Paths to Assess Qualitative Aspects of a System

There are several qualitative aspects of a system such as understandability, maintainability, etc, that cannot be described using a numerical measure. In this section, we provide perspectives on how paths and goals can be used for assessing of a variety of rule base qualities, both internal and external, to facilitate comparison between two systems [Ghezzi et al., 1991; Preece et al., 1993a]. The qualities below are not exhaustive, but are the most popular and frequently cited in the literature [Preece et al., 1993a; Ghezzi et al., 1991; Hamilton et al., 1991; O'Neal and Edwards Jr., 1993; Chen and Suen, 1993; Antoniou and Wachsmuth, 1994]. Note, not all system qualities can be assessed this way. For example, system qualities such as inter-operability, and portability cannot be assessed using paths and goals.

Understandability: The understandability of a system is portrayed to some extent by the individual paths for goal-to-goal progressions and routes to arrive at a solution for permissible initial evidence. The paths and routes thus capture the overall problem solving that occurs. The goal graph extracted from the rule base (section 4.4) allows one to compare the working of a system to that of the mental view of a domain expert: this allows a better understanding of the system, and the fixes required (if any) to obtain the desired behavior.

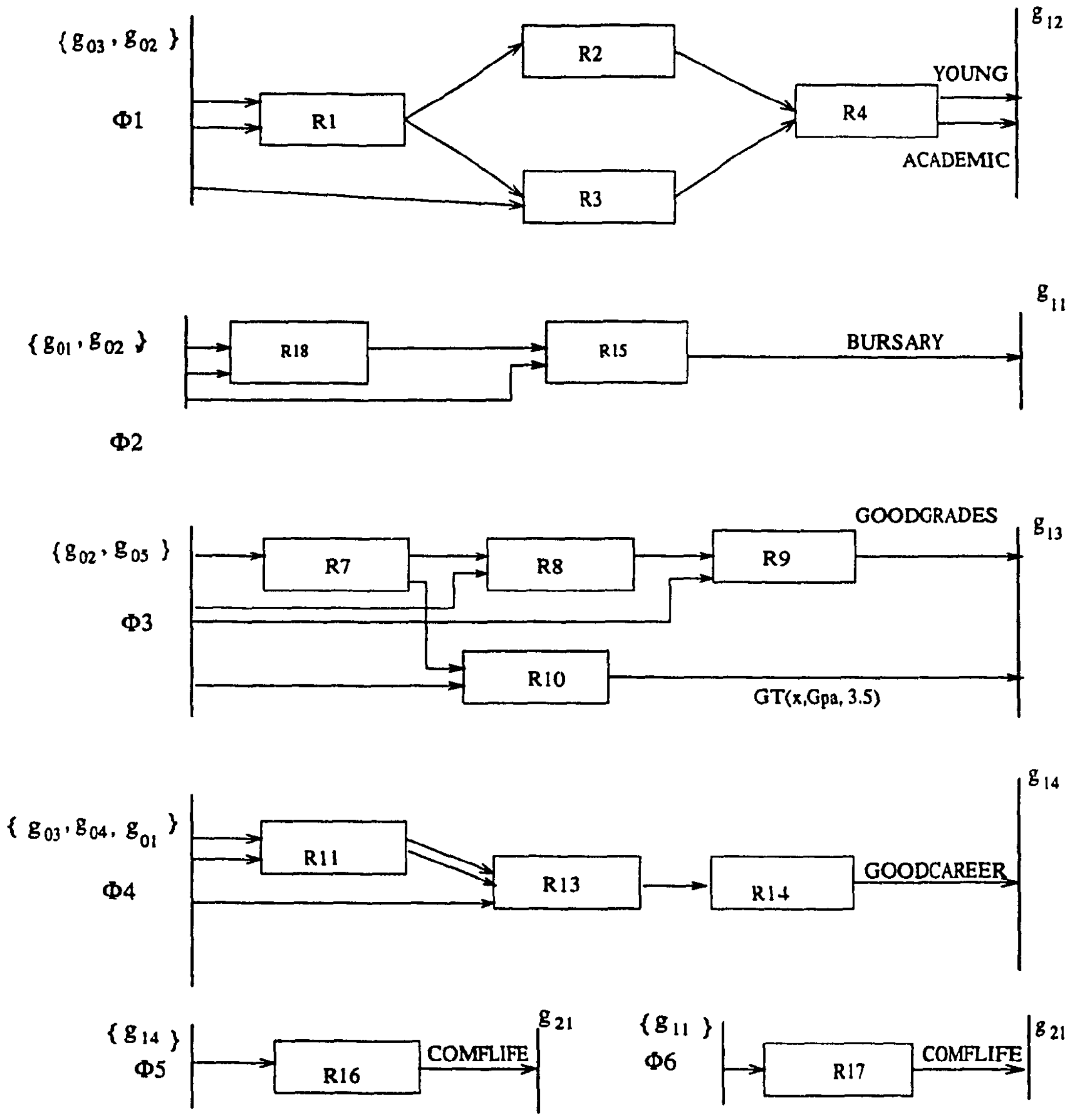
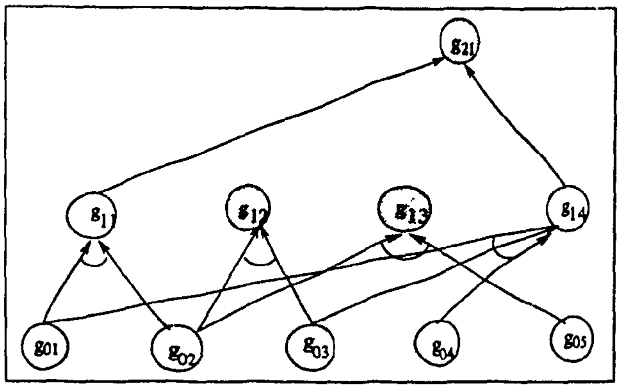


Figure 4.32: *Paths of the rule base shown in Figure 4.31 which is the rule base modified to fix the errors, anomalies and scheme violations detected after evaluation of the rule base shown in Figure 4.7.*



$$
\begin{aligned}
g_{01} &= \text{GOODRECORD}(x, \mathit{Juniorcollege})\\
g_{02} &= \text{REGISTERED}(x)\\
g_{03} &= \text{GREENBORDRID}(x)\\
g_{04} &= \text{HARDWORKING}(x)\\
g_{05} &= \text{DEANSLIST}(x)\\
g_{11} &= \text{BURSARY}(x)\\
g_{12}^{*} &= \text{ACADEMIC}(x) \wedge \text{YOUNG}(x)\\
g_{13}^{*} &= \text{GOODGRADES}(x, y) \wedge\\
&\quad \text{GT}(x, \mathit{Gpa}, 3.5)\\
g_{14} &= \text{GOODCAREER}(x, \mathit{Industry})\\
g_{21}^{*} &= \text{COMFLIFE}(x)
\end{aligned}
$$

Figure 4.33: *The goal graph for the rule base shown in Figure 4.31.*

Modularity: The modularity of a system is described by rule sets, also called rule groups [Jacob and Froscher, 1990], that are required to solve a task, and the relation between the various rule groups. **Path hunter** requires goals to be grouped by tasks. (For small systems, we have just one task: the problem name itself.) The tasks can be treated as portraying the minimally required modularity in the system and the goals of a task as akin to "module interface specification" [Ghezzi et al., 1991; Preece et al., 1993a]: that is, it portrays how one task can be solved using goals only, viewing them as "black boxes." The paths associated with a task refers to rules inferring goals required to solve that task. Thus, this allows grouping of rules associated with that task. The rules that participate in solving more than one task, that is, rules that appear in paths for different tasks serve to connect the tasks, and hence facilitate documenting the relationship between the tasks. Paths for a task and the rules that appear in the paths of more than one task form adequate measures for further modularizing the rule base either by rule re-organization, and/or rule modification as appropriate. As noted by Jacob & Froscher (1990), this can also improve the **maintainability** of the system.

Verifiability: A software system is verifiable if its properties can be verified easily [Ghezzi et al., 1991]. The use of paths and goals to detect the CARD anomalies in a rule base is described in detail in section 4.5.

Complexity: The complexity of rule bases can be assessed using a variety of metric measures such as the cyclomatic complexity [O'Neal and Edwards Jr., 1993; Kiper, 1992]. Paths provide another measure: the mean length of a path, and its "breadth" (the size of the enabling set containing the maximum number of rules in a path). The larger these numbers for a rule base, then greater is the complexity of rule interactions in that rule base because paths localize all the required rule interactions in a goal-to-goal progression.

Chapter 5

Summary and Conclusion

The work done as part of this thesis research and its contributions are summarized. Scope for future research work is briefly described.

5.1 Summary of Work Done

This thesis is a contribution to the evolution of a design framework for rule-based systems consisting of three stages: the functional requirements stage, the design stage and the implementation stage. The functional requirements are abstracted using mileposts for problem solving, called goals. The goals and inviolables specified for a domain, called the goal specification, is intended to capture the essence of problem solving in the domain. Problem solving at this stage is modeled as a sequence of goal inferences, and can be portrayed as an AND/OR graph traversal called the goal graph.

The design stage sets some restrictions to be imposed in mapping the specified goals to the eventual rule base implementation. More specifically, a design scheme is a mapping constraint imposed between a goal and its constituent hypotheses in a rule base. This mapping is non-unique and there are several ways (mappings) of realizing a given goal. This gives rise to different design schemes. The different design schemes provide flexibility in choosing a scheme to best suit the system development criteria. The use of goal-based design schemes is not only useful for development, but can also be used to restructure and optimize an existing rule base. This was outlined by means of a case study.

The mapping restrictions imposed by the various design schemes are not mutually exclusive: some scheme restrictions are more general than others. Based upon the generality of the scheme mapping restrictions, a specific relationship between the design schemes, called inheritance, was analyzed in this thesis. A design scheme $\mathcal{D}_B$ inherits from another scheme $\mathcal{D}_A$, whenever every rule base $\mathcal{R}_A$ that adheres to $\mathcal{D}_A$ also adheres to $\mathcal{D}_B$, but the converse need not be true. Inheritance between the design schemes is an important notion because it can allow for certain compromises between development and maintenance. This was demonstrated by a cost-benefit analysis of the maintenance operations between two scheme $\mathcal{D}_A$ and $\mathcal{D}_B$ that have an inheritance relationship between them. In addition, the mapping restriction imposed by a design scheme can be analyzed to provide an assessment of certain system qualities in an implementation independent way. Metric measures were developed as a part of this thesis based upon the restrictions imposed by a design scheme, to facilitate comparison between design schemes to help choose one for a domain. The values of these metrics are influenced by scheme inheritance.

The compromises between development and maintenance, and for quality improvement in this design framework for rule-based systems is possible by the existence of a rule base transformation from a rule base adhering to scheme $\mathcal{D}_B$ to a rule base adhering to scheme $\mathcal{D}_A$, whenever the former inherits from the latter. The transformation preserves the rule base structure, content, runtime inference, and problem solving semantics (goal types). The details of the rule base transformation, using a sample pair of design schemes, was outlined with an example.

The criteria to choose a design scheme from a set of schemes for a domain are not unique. We provided a set of empirical criteria that can be used to select or reject a design scheme for a domain as an aid to system developers. Such analyses are meant to provide a developer with a deeper understanding of the goal-to-hypothesis mapping restrictions, scheme relationships, and on the issues in system development, evaluation, and maintenance. This should facilitate choosing a scheme for a domain.

A rule base constructed based on a given goal specification, and adhering to a specific design scheme implements the problem solving by rule sequences that progress from goal(s) to goal from the given goal specification. Every such rule sequence in the rule base is said to have realized a connector in the goal graph; thus, there may be more than one rule sequence to realize a given connector. These rule sequences are called paths. A rule graph can be used to pictorially depict the accessibility of the rules in a path.

The extent to which a given rule base realizes the acquired knowledge of goal inference is represented by the paths in the rule base; they are collectively said to portray the structure of the rule base. The extraction of paths and the goal graph from a rule base is called structure extraction, and it influences a variety of evaluation processes for rule-based systems. However, the extraction of paths from a rule base using a given goal specification is a non-trivial problem because procedures that extract inference chains from a rule base have an exponential complexity in the worst case. The use of goal specification to control the computation required to extract paths was described. The use of a tool called **path hunter**, to extract the paths in a rule base from a given goal specification, and the feasibility of this model to handle large rule-based systems were also described [Grossner et al., 1993].

All the proposed evaluation methods are based on the structural model of a rule base (definition 14, page 83); thus, evaluation is based upon paths and goals rather than individual rules. The advantage of this approach is that paths automatically account for the rule interactions that take place in the rule base. Evaluation that does not take into account the rule interactions that take place may miss some errors and/or anomalies that are present. The system evaluation methods proposed in this thesis allow a general measurement of performance, structurally validate the system, and perform verification to detect rule base anomalies. The individual evaluation methods are summarized below.

Structural Validation The structural validation of a system is conducted by extracting the paths in its rule base and checking if the goal-to-goal progression indeed captures the mental view of a domain expert from whom this knowledge was acquired. In addition, selected test cases are applied and the rule traces are mapped to paths to measure the coverage of the system. This also provides a performance measure albeit it is more specific to the test suite used. The tool **path tracer** described in this thesis can be used to measure the path coverage of a system. The path statistics obtained as part of system validation can be used to assess a variety of rule base qualities.

Performance Evaluation The performance of a system can be measured using traditional benchmarking, but general performance measures are desired when considering systems operating in critical domains [Giovanni, 1989]. Performance evaluation is based on measuring the usefulness of a goal in problem solving, called goal relevancy. The goal relevancy and relevancy ratio of a system can be measured by detecting unreachable final goals and "dead end" intermediate goals in the rule base.

Verification The verification of a rule base can be performed by identifying certain rule situations that can be indicative of the CARD anomalies. They are called rule aberrations because they indicate an abnormality in the system. Goal specification, paths, and the goal graph extracted from the rule base allows for a comprehensive detection of the CARD anomalies in a rule base at any stage during its construction: simply spot paths that adhere to one or more rule aberrations. The details of the rule aberrations to facilitate the detection of the CARD anomalies in a rule base were described.

5.2 Summary of Contributions

The contributions of this thesis are summarized below:

1. **The goal specification approach to abstract problem solving.**

   Abstracting domain knowledge in terms of goals and relationships between goals presented in this thesis enables a knowledge engineer to succinctly capture the important aspects of problem solving in the form of a goal graph. The functional view of problem solving represented by a goal graph can be useful for empirical evaluation because the correctness of a system can be confirmed by checking the goal graph extracted from its rule base for agreement with the mental view held by a domain expert in solving problems [Preece, 1990; Vinze, 1992]. The goal specification approach is not only useful for developing a new system, but also for restructuring an existing system. This was demonstrated by means of a case study (chapter 2, section 2.3).

2. **The design schemes for rule-based systems.**

   The design stage, as viewed in this thesis, is intended to suit the characteristics of a wide range of domains. The various design schemes presented impose different restrictions in mapping goals to hypotheses. The analysis developed as part of the design stage through an examination of the relationships between the design schemes portray the compromises between development and maintenance in the design framework (chapter 3, section 3.3). By carefully identifying and encoding the goals of a domain, a system can be designed in a better way: this aspect was demonstrated by the re-design of the library expert system (chapter 3, section 3.2). We do not specifically recommend any one design scheme because different application domains exhibit different characteristics that should be taken into account when choosing a scheme. In this thesis, we compared the design schemes based upon one aspect: their ease in system development versus system maintenance. This could serve as a guide to system developers in choosing a scheme.

3. **The implementation model of a rule base to capture rule interactions.**

   An implemented rule base adhering to a design scheme can be analyzed in terms of rule sequences, called paths, that infer the specified goals. This is called the structural model of a rule base (chapter 4, section 4.1). Every path in a rule base maps to a definite portion of the acquired knowledge, thus explicating how that knowledge is manipulated in the system.

   It has been observed in the literature that the understandability of a rule base is often obscured by the interactions that occur between its rules. The path concept localizes the rule interactions that occur in a given goal-to-goal progression. Thus, a developer can inspect the paths extracted from an implemented rule base to check if a given sequence of rules inferring a goal indeed represents the rule interaction that was intended. Such an inspection helped to uncover some rule base coding errors in the Blackbox expert system [Grossner et al., 1993; Preece et al., 1993b]. In addition, the structural model of a rule base is useful for the evaluation of rule bases.

4. **The implementation model and its role in system evaluation.**

   In this thesis, we have viewed the system evaluation to comprise of verification for detecting CARD (circularity, ambivalence, redundancy, and deficiency) anomalies, validation for ensuring a system's compliance with its requirements, and performance and quality assessment for assessing its other attributes. The various procedures for evaluating rule-based systems based upon paths and goals encompass the above evaluations: verification can be performed by detecting rule aberrations, validation by path extraction and path coverage measurement, and performance and quality assessment can be done by measuring the goal relevancy, the relevancy ratio, and other path characteristics of the system. The tools, algorithms, and results from case studies reported in this thesis illustrate the practicability of these procedures.

In summary, the feasibility of the proposed design framework for reverse engineering (restructuring for optimization), development, and evaluation strengthens the practicability of this approach. The case studies portray the applicability of this framework for managing rule-based systems consisting of hundreds of rules: for example, modularization, restructuring, and computation control for path extraction.

Overall, the proposed design framework provides a systematic and rigorous methodology for the development of rule-based systems. The design schemes are pragmatic for both development and reverse engineering, and also provide a strong link between system conception and implementation. The formalizations developed for modeling rule base design (design schemes, quantification using metrics, inheritance, and transformation), and rule base structure (paths and their properties) provide a strong analytical foundation for this framework. The applicability of the design schemes for development and for reverse engineering. automated rule base transformation to facilitate maintenance, and tools and algorithms to facilitate system evaluation underscore the practicality of this approach.

5.3 Scope for Future Research

This work opens up a lot of exciting avenues for future research in the design and development of not only stand alone rule-based systems, but also multi-agent systems, where the individual agents are implemented as rule-based systems.

In the short term, two improvements can be made to the existing formalism for rule bases to further improve its scope.

1. The first enhancement would be to provide a simple semantics that allows goals and rules to contain negated atoms. For simplicity, a closed-world assumption stated informally as "negation of an atom is its absence from working memory" can be used initially (this negation semantics is used in CLIPS). Thus, the formalism for a path in a rule base should also accommodate for negated atoms in rule antecedents and consequents. Currently, the rule splitter (section 4.3.1) has provisions to hold negated atoms in a rule, but they are left empty. All that is required is to fill this data structure appropriately when the rule base is parsed, and augment the way rule accessibility is computed.
2. The second enhancement would be re-write the **path hunter** and **path tracer** tools in a language, such as C, efficiently. This would improve their portability.

The algorithms discussed for measuring goal relevancy, and for detecting the rule aberrations should be implemented as part of **path hunter**. Currently, **path hunter** provides limited verification support by flagging rule circularities and rules that do not appear in any path, and by recording path statistics as part of path extraction. However, in the interest of modularity, a separate tool to augment the existing tool suite is desirable.

A major enhancement to the goal formalism is the use of a general first order logic formula to represent a goal. Thus, a *language* is required to express goals and constraints of a domain. The following question, however, arises: how does the formalism to characterize a path in a rule base would change? The use, feasibility, and practicability of this extension should be explored. Note, the additional expressive power to represent goals can be used for modeling a wide range of problems in a domain.

In this thesis, the use of inheritance for automated rule base transformation was explored. A rule base $\mathcal{R}_B$ adhering to a scheme $\mathcal{D}_B$ can be automatically transformed into a rule base $\mathcal{R}_A$ adhering to another scheme using one of the transformation procedures. An interesting application of inheritance is to view the transformation process in reverse: is it possible to *convert* a rule base $\mathcal{R}_A$ adhering to a scheme $\mathcal{D}_A$ into another scheme $\mathcal{D}_B$ that inherits from $\mathcal{D}_A$, such that the resulting rule base is more efficient than $\mathcal{R}_A$? In other words, the use of inheritance for automated performance optimization of rule bases would enhance the use, and the application of the framework discussed in the thesis.

An interesting aspect of the goal graph based abstraction of problem solving is its potential for explanation: providing a justification (usually to the user) of the system's conclusion from a given set of initial evidence. This justification can be provided in terms of the justification associated with the inferred intermediate goals that were subsequently used to infer the final goals. The goal graph in this case can be viewed as a semantic network, and the explanation can be easily given by keeping track of the set of routes that were traversed by the system during problem solving.

A preliminary exploration of the use of paths to analyze the performance of multi-agent systems by examining the data distributed among them appears in [Grossner, Gokulchander, Preece, and Radhakrishnan, 1993; Grossner, Preece, Gokulchander, Radhakrishnan, and Newborn, 1994]. The results of using paths to capture the data distribution in a multi-agent environment for performance analysis provides motivation to extend this design framework for multi-agent systems. However, several issues arise in so doing. Some of them are outlined below:

1. *The agent modeling issue:* Should an agent be modeled as a set of paths, or should every path be modeled as a ("pseudo") agent? In particular, note that if the goal and path formalism are extended to include negated atoms, then the closed-world assumption cannot be used in a multi-agent setting. For example, if an agent, say X removes an atom A from its working memory, then $\neg A$ is true in X by closed-world assumption. But, if another agent, say Y infers atom A, then A is true in Y (conversely, $\neg A$ is false in Y). But, now the system as a whole is inconsistent: considering the "worlds" of agents X and Y together, we have an inconsistency because both A and $\neg A$ are true.

2. *The design issue:* What "design schemes" are possible for a multi-agent setting? One way would be to restrict ourselves with the existing schemes, and assume that every agent adheres to a scheme.

3. *The evaluation issue:* How should the evaluation procedures be extended to

handle multi-agent evaluation? The extensions are, unfortunately, non-trivial. For example, consider the notion of goal relevancy: Suppose if a goal g is irrelevant in agent X, but is relevant in agent Y, what can we say about goal g? (If goal g is not realized in agent X, then the issue is trivial, but if it is also realized in agent X, the issue requires a deeper analysis.)

4. *The problem solving model issue:* In this approach for single rule-based systems, problem solving can be modeled as a goal graph (chapter 2, section 2.2). However, in a multi-agent setting, if goals are realized in different agents, the question arises: how to extract the goal graph of the system?

Appendix: Algorithms to Detect Rule Aberrations

It is assumed that the paths are pre-processed into a set of indices before the aberrations can be spotted. The typical indices are the rule index (list of paths, and routes in which a rule appears), and fact index (list of paths and routes in which the fact appears). Whenever a rule, or an atom is flagged, an appropriate field is set in the indices. This facilitates later checking for flagged rules.

An algorithm to detect aberration 1 (page 117) is given below. The route computation part of the algorithm requires that the algorithm for goal graph extraction to determine relevant and irrelevant goals has been run to extract all the relevant and irrelevant routes. The route extraction procedure `GetRoutes` is repeated below for convenience. The rest of the algorithm is straight forward: simply check for rules that does not appear in any route, or appear in only irrelevant routes. Note, in the algorithm for aberration 1, we have made use of a boolean function *Irrelevant*(x) that should return *true*, if x is an irrelevant route.

```
Procedure GetRoutes;
Input: Paths, Goal specification ($F$ is the set of final goals,
and $I$ is the set of intermediate goals)
Output: The set $\gamma$ of all routes in the goal graph
begin
    1. $\gamma := \emptyset$
    2. Mark every intermediate goal in $I$ as *unvisited.*
    3. Select a $f \in F$; mark every path inferring $f$ as *unseen*
    4. Repeat
            4.1 While (one more path $\Phi$ to $f$ is marked unseen) do
                    4.1.1. $I' := \emptyset$
                    4.1.2. Pick the path $\Phi$ inferring $f$ marked unseen;
                    mark this connector as *seen*
                     /* obtain the goals at the other end of $\Phi$ */
                    4.1.3. Derive the subset $G$ of $I$ that contribute to the start
                    state of path $\Phi$.
                    4.1.4. $I' := I' \cup G$  /* $G$ obtained in previous step */
                    and mark intermediate goals in $I'$ as *visited.*
                     /* Traverse the goal graph using the connectors */
                    4.1.5. Apply 4.1.3—4.1.4 recursively for all paths to each
                    goal in $I'$ until one of the following is true:
                            If a level-0 goal is reached, then
                              Mark this sequence of paths as a route $\Pi_f$
                            else Mark this sequence of path as an irrelevant
                                route $\Pi_f$ culminating on final goal $f$
                     /* Accumulate the routes collected so far into $\gamma$ */
                    4.1.6. $\gamma := \gamma \cup \Pi_f$
            end  /* while */
             /* All possible traversals from $f$ has been completed */
    Until (as long as there is a goal in $F$ to which step 3.1 has not been applied)
    5. For every intermediate goal $i$ not *visited* (as determined by step 4),
            5.1. Repeat step 3.1 to enumerate the irrelevant routes $\Pi_i$ culminating on $i$
            5.2. $\gamma := \gamma \cup \Pi_i$
    6. Output $\gamma$
end.  /* End of GetRoutes */

Procedure aberl; /* detect aberration 1 */
Input: Rule index (RI), Goal specification.
begin
```

```
  GetRoutes; /* get the set $\gamma$ */
  For all rules $r \in RI$
    if $(\forall \Pi \in \gamma)\ r \in \Pi \wedge Irrelevant(\Pi)$
      Flag rule $r$ /* $r$ appears only in irrelevant routes */
    if $(\forall \Pi \in \gamma)\ r \notin \Pi$
      Flag rule $r$ /* $r$ appears in no route */
end.
```

An algorithm for detecting aberration 2 (page 117) to detect redundant rule chains appears below. The function $Subsume(X, Y)$ returns true whenever there is a subsumption relationship between X and Y. We have also used a function $Goals(\Phi)$ that returns the set of goals required by a path Φ. Note the use a break statement in order to exit from a nested loop (this semantics is similar to the break statement in C) for convenience. For efficiency, we restrict ourselves to only paths in relevant routes in step 2 because paths appearing in only irrelevant routes are flagged by step 1.

```
Procedure aber2; /* detect aberration 2 */
Input: Rule Index, Paths, Goals, The set $\gamma$.
begin
  /* Flag every path that appears in an irrelevant route */
  1. For every path $\Phi$, do
    1.1. redundant_chain := false;
    1.2. If $(\forall \Pi \in \gamma)\ \Phi \in \Pi \Rightarrow Irrelevant(\Pi)$, then
              redundant_chain := true;
      else
              redundant_chain := false; /* $\Phi$ is in at least one relevant route */
              break; /* break out of inner for loop */
    1.3. If redundant_chain = true, then
        Flag path (and rules in) $\Phi$. /* path appears only in irrelevant routes */
  /* Flag redundant rule chain pairs based on subsumption */
  2. For every path $\Phi_i$, do
```

```
2.1 For every path $\Phi_j$, $j \neq i$, do
  2.1.1. If $(\exists \Pi_1, \Pi_2 \in \gamma)\ \Phi_i \in \Pi_1 \wedge \Phi_j \in \Pi_2 \wedge Relevant(\Pi_1) \wedge Relevant(\Pi_2)$, then
    2.1.1.1. $goal\_atoms\_1 := true$;
    2.1.1.2. $goal\_atoms\_2 := true$;
    2.1.1.3. For every $g \in Goals(\Phi_i)$, do
      $goal\_atoms\_1 := goal\_atoms\_1 \wedge g$;
    2.1.1.4. For every $g \in Goals(\Phi_j)$, do
      $goal\_atoms\_2 := goal\_atoms\_2 \wedge g$;
    /* ensure input goal subsumption */
    2.1.1.5. If $Subsume(goal\_atoms\_1, goal\_atoms\_2) = true$, then
      2.1.1.5.1. If $Subsume(g_1, g_2) \wedge \Phi_i \vdash g_1 \wedge \Phi_j \vdash g_2$, then
        Flag $\Phi_i$ and $\Phi_j$ /* redundant rule chains */
        Flag every rule $r$ appearing in only $\Phi_i$ and/or $\Phi_j$
    /* An additional perspective for non-goal atoms; optional. */
        If a non-goal atom $a$ is inferred in both $\Phi_i$ and $\Phi_j$, then
          Flag this (sub)sequence of rules in $\Phi_i$ and $\Phi_j$
end.
```

In the algorithm for aberration 3 (page 118) to detect redundant atoms below, the function $InferenceNo(h, \Phi)$ associated with atom h is assumed to give the number of times h is inferred in path Φ. The $Antecedent(r)$ ($Consequent(r)$) is assumed to return the list of atoms in the antecedent (consequent) of a rule r. Note, if the rule index also contains a boolean field that evaluates to true if a rule r is flagged redundant, then step 8.3 can set this field, and step 8.4 can be done after all the rules in the rule index are processed. This can speed up the output processing to some extent.

```
Procedure aber3; /* detect aberration 3 */
Input: Rule Index (RI), Fact Index (FI), Paths .
  1. $M := \emptyset$ /* to hold multiply inferred goal atoms */
  2. $D := \emptyset$ /* to hold dangling non-goal atoms */
  3. For every $h \in FI$, do
```

```
    3.1. multiply_inferred := true;
    3.2. For every path $\Phi$ such that $\Phi \vdash h$,
        3.2.1. if $GoalAtom(h) \wedge InferenceNo(h, \Phi) \leq 1$
            multiply_inferred := false;
    3.3. If multiply_inferred = true, then
        3.3.1. $M := M \cup h$
4. Flag atoms in $M$ /* atoms multiply inferred in all path they appear */
5. If $(\exists r \in RI)\ Consequent(r) \subseteq M$ (from step 2), then
        Flag rule $r$ /* redundant toe rule */
6. For every $h \in FI$, do
    6.1 dangling := false;
    6.2. For every path $\Phi$ such that $\Phi \vdash h$,
        6.2.1. if $NonGoalAtom(h) \wedge (\nexists r \in \Phi) h \in Antecedent(r)$
            dangling := true; /* $h$ is a dangling atom in $\Phi$ */
    6.3. If dangling = true, then
        6.3.2. $D := D \cup h$
7. Flag atoms in $D$ /* atoms dangling in every path inferring them */
8. For every rule $r \in RI$ such that $Consequent(r) \cap D \neq \emptyset$, do
    8.1. redundant1 := false;
    8.2. redundant2 := false;
/* Flag a rule whose consequent is either dangling or inferred by other rules in a path */
    8.3. For every path $\Phi$ where $r \in \Phi$,
        8.3.1. For every $h \in Consequent(r)$,
            If $h \notin Consequent(r) \cap D \Rightarrow$
                    $(\exists r' \in \Phi, r \neq r') h \in Consequent(r')$, then
                redundant1 := true;
            If $h \in Consequent(r) \cap D$, then
                redundant2 := true;
    8.4. If redundant1 $\wedge$ redundant2, then
        Flag rule $r$.
```

end.

Aberration 4 (page 120) is the rule aberration aberration that is applicable whenever a rule r can replace another rule r' in every path where r' appears, or vice versa. We repeat the conditions from the text (page 119) for convenience. Let $r_1 : A_1 \wedge A_3 \rightarrow H$ and $r_2 : A_1 \wedge A_2 \rightarrow H'$ be any two rules such that A_3 is a goal atom, the remaining atoms are non-goal, and the H's represent hypotheses which can be a conjunction of atoms. Further, let H subsume H'. Then, the rule pair $< r_1, r_2 >$ is flagged redundant based upon the following conditions:

(i) If every path in which r_1 appears has at least one rule that infers A_2 (in other words, whenever r_1 fires, r_2 can fire) then perhaps r_1, or r_2 is redundant.

(ii) Whenever r_2 occurs in a path from G to g and A_3 is contained in G (in other words, r_1 can be used to traverse between any G to g whenever r_2 can do so), then perhaps r_1, or r_2 is redundant.

For example, (1) if H is $BIRD(x)$ and H' is $BIRD(Tweety)$, and condition (i) above holds then r_2 is redundant; and (2) if H is $BIRD(x)$ and H' is $BIRD(x) \wedge SINGS(x)$, and condition (ii) above holds then r_1 is redundant.

The details of detecting aberration 4 for rule subsumption appears below. Note, we the use of notation RI(r_i) to indicate the index information associated with rule r_i, and the use of BREAK statement to break out of *any* level of loop nesting for ease in readability. The function *NonGoalAtom* (*GoalAtom*) should return a list of non goal atoms (goal atoms) from the list passed as its argument.

```
Procedure aber4; /* detect aberration 4 */
Input: Rule Index (RI), Fact Index (FI), Paths.
  1. For all rule pairs $r_1, r_2$ do /* a two loop construct */
  /* get list of non-goal atoms from the antecedents */
    1.1. $N_1 = NonGoalAtom(Antecedent(r_1))$;
    1.2. $N_2 = NonGoalAtom(Antecedent(r_2))$;
  /* Get the goal atoms from the antecedents */
    1.3. $G_1 = GoalAtom(Antecedent(r_1))$;
    1.4. $G_2 = GoalAtom(Antecedent(r_2))$;
  /* Check for subsumption conditions */
    1.6. contained1 := true;
    1.7. For every path $\Phi$ in RI($r_1$), do
        1.7.1 If $G_2 \notin Goals(\Phi)$, then
            1.7.1.1. contained1 := false;
            1.7.1.2. break;
  /* ensure non-goal atoms in antecedent of $r_2$, but not in $r_1$
  is supplied by some rules in every path where $r_1$ appears. This ensures
  that $r_2$ can replace $r_1$ in those paths */
    1.8. For every $a \in N_2$ such that $a \notin N_1$, do
        1.8.1. NonGoalInference := true;
        1.8.2. For every path $\Phi$ in RI($r_1$), do
            1.8.2.1. If $NOT(\Phi \vdash a)$, then
                1.8.2.1.1. NonGoalInference := false;
                1.8.2.1.2. BREAK; /* Out of outer for loop as well */
  /* Finally, put all together to check for subsumption */
  2. If contained1 $\wedge$ NonGoalInference $\wedge$
        $Subsume(Consequent(r_1), Consequent(r_2))$, then
    2.1. Flag rule pairs $r_1$ and $r_2$.
  /* Check for syntactic subsumption; this ensures compatibility with
  existing CARD detection tools */
  3. For all rule pairs $r_1, r_2$ in RI, do /* a two loop construct */
    3.1 If $Susbume(Antecedent(r_1), Antecedent(r_2))$, then
        3.1.1. Flag rule pairs $r_1$ and $r_2$.
end.
```

An algorithm to detect aberration 5 (page 121) appears below. It requires only the current set of flagged rules and atoms.

```
Procedure aber5; /* detect aberration 5 */
Input: Rule Index (RI), Fact Index (FI).
```

```
begin
  Let FlaggedR := current set of flagged rules from RI;
  Let FlaggedA := current set of flagged atoms from FI;
  For all $x$ such that $x \in FlaggedA$, do
    If $\exists r \notin FlaggedR \wedge x \in Consequent(r)$, then
      Flag rule $r$ /* $r$ potentially makes a useless inference */
end.
```

An algorithm to detect aberration 6 (page 121) appears below. It requires only the current set of flagged rules and atoms.

```
Procedure aber6; /* detect aberration 6 */
Input: Rule Index (RI), Fact Index (FI).
begin
  Let FlaggedR := current set of flagged rules from RI;
  Let FlaggedA := current set of flagged atoms from FI;
  For all $x$ such that $x \in FlaggedA$, do
    If $\exists r \notin FlaggedR \wedge x \in Antecedent(r)$, then
      Flag rule $r$ /* $r$ can become potentially unreachable */
end.
```

The algorithms to detect ambivalence (aberrations 7, 8 and 9, page 124) appears below.

```
Procedure aber7; /* detect aberration 7 */
Input: Fact Index (FI), Set of Inviolables (S), Goal Specification ($\mathcal{G}$).
begin
  1. For every path $\Phi$ do
    1.1. Let $NGA$ := The set of set non goal atoms in a path $\Phi$;
      1.1.1. If $\exists x \in S \wedge Subsume(x, NGA)$, then
        Flag path $\Phi$ as violating inviolable $x$ (by inferred non-goal atoms).
    1.2. Let $GA$ := The set of goal atoms in path $\Phi$;
      1.2.1. If $\exists x \in S \wedge Susbume(x, GA)$, then
      Flag path $\Phi$ as violating inviolable $x$ (by inferred goal atoms).
     /* Also check if all atoms in a path ($NGA \cup GA$) above is
     subsumed by an inviolable. */
  2. For every goal $g \in \mathcal{G}$, do
    2.1. If $\exists x \in S \wedge Susbume(x, g)$, then
     Flag goal $g$.
  3. If $\exists x \in S \wedge Susbume(x, Goals(\Phi))$, then
    Flag $\Phi$ as requiring an inviolable to start.
end.

Procedure aber8; /* detect aberration 8 */
Input: Fact Index (FI), Set of Inviolables (S).
begin
  1. For every route $R$, do
    1.1. Let $NGA$ := The set of set non goal atoms in route $R$;
     1.1.1. If $\exists x \in S \wedge Subsume(x, NGA)$, then
      Flag route $R$ as violating inviolable $x$ (by inferred non-goal atoms).
    1.2. Let $GA$ := The set of goal atoms in route $R$;
     1.2.1. If $\exists x \in S \wedge Susbume(x, GA)$, then
      Flag route $R$ as violating inviolable $x$ (by inferred goal atoms).
     /* Also check if all atoms in a route ($NGA \cup GA$) above
     is subsumed by an inviolable. */
end.

Procedure aber9; /* detect aberration 9 */
Input: Fact Index (FI), Set of Inviolables (S), The set $\gamma$, Goal Specification ($\mathcal{G}$).
begin
  1. For every route $R$, do
    1.1. Let $GA$ := The set of level-0 goal atoms of route $R$;
     1.1.1. If $GA \notin \mathcal{G} \vee (\exists x \in S) Subsume(x, GA)$, then,
      Flag route $R$.
  /* The next step of this algorithm requires tracing
  routes for test cases that deliberately contain inviolables
  and/or impermissible combinations of initial evidence (see page 124) */
end.
```

The algorithm to detect path circularity (aberration 10, page 126) is given below. If the goal graph extracted from the rule base is represented so that every goal g is ordered using the form:

[g [Goals on AND edges incident on g] [Goals on OR edges incident on g]]

then, the algorithm to detect circularities in the goal graph is similar to the way rule accesibilities are computed in path hunter which also checks for rule circularites when enumerating paths (section 4.3.1). The code, with minimal changes, can be re-used to check for path circularity.

```
Procedure aber10; /* detect aberration 10 */
Input: Goal graph G (assumed ordered as discussed above).
begin
  1. For every goal $g \in G$, do
    1.1. Mark goal $g$ as seen.
    1.2. Let $S'$ := Set of goal required to infer $g$
    1.3. Compute $S'$ for every goal $g'$ in $S'$ recursively after
    marking $g'$ as seen.
      1.3.1. If a goal that was seen before needs to be included in the current set $S'$, then
        flag the set of goals seen as involved in a circularity.
/* The above method is straight forward. For efficiency, any cycle checking algorithm
for graphs can be used */
end.
```

Glossary

1. **Accessibility of a Rule** A rule r_2 is accessible from another rule r_1 iff a non-goal atom in the antecedent of r_2 is unifiable with an atom in the consequent of r_1.

2. **Accessibility Set of a Rule** The set of all rules from which a rule r is accessible is called the accessibility set of r.

3. **Adhering to a Design Scheme** Given a goal specification $\mathcal{G}$, a rule base $\mathcal{R}$ is said to adhere to a design scheme $\mathcal{D}$, iff every intermediate (final) goal realized in $\mathcal{R}$ satisfies the intermediate (final) goal mapping restriction imposed by $\mathcal{D}$.

4. **Adhocness of a Goal g in a Scheme $\mathcal{D}$** The adhocness of a goal g in scheme D is the ratio of the maximum number of non-corresponding constructs allowable in g to the least number of corresponding constructs allowable in g without violating the restrictions of scheme $\mathcal{D}$.

5. **Attainability of a rule** A rule r_2 is attainable from another rule r_1 iff an atom in the antecedent of r_2 is unifiable with an atom in the consequent of r_1.

6. **Behavior** The behavior of a rule-based system is the set of rules fired and hypotheses inferred at run time for a given input. In the literature, the terms 'run time behavior', and 'problem solving behavior' are also used synonymously when referring to this term.

7. **Connector** An AND edge, or an OR edge in a goal graph.

8. **Consumption of a Rule** A rule r in a path is fully consumed iff every non-goal atom in the consequent of r unifies with an atom in the antecedent of

one other rule r' in the same path. A rule is partly consumed iff it is not fully consumed.

9. **Design Scheme** A design scheme is a mapping restriction imposed in realizing intermediate and final goals in a rule base.

10. **Discrepancy in a Transformation** When transforming a rule base adhering to scheme $\mathcal{D}_B$ into one that adheres to scheme $\mathcal{D}_A$ by converting hypothesis types in a goal in order to annul a design restriction in scheme $\mathcal{D}_A$. the transformation can inadvertently violate another restriction of that scheme. When this occurs, the transformation is said to have reached a discrepancy.

11. **Enabled Rule** A rule whose antecedent is true.

12. **Enabling Set of a Rule** The minimal set of rules whose firing enables a rule r.

13. **Fact** A generic term referring to an atom, or a predicate.

14. **Flagging** The identification of specific rules and atoms that can be causal for some anomalies in a rule base.

15. **Goal** An abstraction of a state in problem solving.

16. **Goal and Non-goal Atoms** The atoms that are present in goals are called goal atoms. The other atoms (used for rule encoding) are called non-goal atoms.

17. **Goal Distance Metric** The distance between the conception and the realization of a specified goal g is defined as the number of non-corresponding constructs used for realizing the goal g in the rule base.

18. **Goal Graph** A graphical way of portraying problem solving in the domain using goals.

19. **Goal Specification** The set of goals and inviolables specified for a domain.

20. **Inference Engine** A mechanism that controls rule execution in a rule-based system.

21. **Intermediate and Final Goals** Goals that are inferred in order to facilitate reaching a solution are called intermediate goals. The goals that are used for

indicating domain solutions are called final goals.

22. **Non-corresponding Construct in a Goal** A final hypothesis used in the realization of an intermediate goal (or, vice versa) is said to be a non-corresponding construct in that goal.

23. **Path** A path in a rule base is sequence of rules that infer a goal from a given set of goals. It represents a connector.

24. **Path Coverage** Measuring the extent the paths in a rule base are exercised for a given test suite.

25. **Path Hunter** A tool to extract paths from a rule base.

26. **Path Tracer** A tool to measure path coverage.

27. **Pattern Matching** The computation required to check if the antecedent of a rule is satisfied.

28. **Realizing a Goal** A set of rules ρ in the rule base is said to realize a given goal $g = A_1 \wedge A_2 \wedge A_n$, iff the consequent of these rules collectively contain the atoms $A_1, A_2, \ldots, A_n$.

29. **Relevant and Irrelevant Goals** Given the initial evidence corresponding to a problem, if the system infers some final goals that correspond to the solution of the problem, then the inferred final goals are said to be relevant. A final (intermediate) goal is irrelevant iff it is not a relevant final (intermediate) goal.

30. **Relevancy Ratio** The relevancy ratio of intermediate goals is defined as $\frac{NI_r}{NI_t}$ where, NI_r is the number of relevant intermediate goals and NI_t is the total number of intermediate goals in a goal specification. The relevancy ratio of final goals is defined as $\frac{NF_r}{NF_t}$ where, NF_r is the number of relevant final goals and NF_t is the total number of final goals in a goal specification.

31. **Relevant and Irrelevant Routes** A route from a level-0 goal to a solution is called a relevant route; any other route from a level-n goal to a goal g', where $n \geq 1$ and g' is not a solution, is called an irrelevant route.

32. **Route** A sequence of one or more paths.

33. **Rule Aberration** A rule aberration in a rule base consists of a set of paths that portray the manifestation of one or more rule base anomalies.

34. **Rule Base Distance Metric** The distance metric of a rule base is represented using the largest distance metric of the interm.diate and final goals that are realized in the rule base. It is represented as an ordered pair $\Delta =< \delta_i, \delta_f >$ where, δ_i (δ_f) is the largest distance metric of the intermediate (final) goals realized in the rule base.

35. **Rule Base Structure** The set of paths in a rule base adhering to a design scheme.

36. **Rule Base Transformation** An automated way of re-writing rules in a rule base $\mathcal{R}_A$ adhering to a scheme $\mathcal{D}_B$ so that it adheres to another scheme $\mathcal{D}_A$, where $\mathcal{D}_B$ inherits from $\mathcal{D}_A$.

37. **Ru'e Equivalence Class** The set of split rules (produced by the rule splitter module of path hunter) that are identical are said to form a rule equivalence class.

38. **Rule Graph** A graphical representation of the rule accessibility relationship in a rule base.

39. **Rule Interactions** The effect of the causality relationship between the rules for rule firing during problem solving.

40. **Salience** A form of rule priority assignment used in CLIPS.

41. **Scheme Inheritance** A design $\mathcal{D}_B$ is said to inherit from design $\mathcal{D}_A$, iff every rule base $\mathcal{R}_A$ adhering to scheme $\mathcal{D}_A$ also adheres to scheme $\mathcal{D}_B$.

42. **Shaved Path** A path containing only fully consumed rules. Otherwise, the path is said to be unshaved.

43. **Test Case** A set of initial evidence.

44. **Test Suite** A set of test cases.

45. **Working Memory** A component of a rule-based system used for storing the data produced by rule firings at run time.

Notation Summary

Symbol	Represented Concept
$\mathcal{G}, \gamma$	Set of goals.
H	Set of hypotheses in a rule base.
P	A problem in a domain.
$\mathcal{A}$	Anomaly set of a rule base.
$\mathcal{C}$	Causality relationship (between rules in a rule base).
$\mathcal{D}$ and μ	Design scheme and mapping respectively.
<Fn, Im>	Denotes a specific design scheme; numbers 'n' and 'm' vary from 1 to 5.
$\mathcal{G}$, I, and F	Goal Specification, Set of intermediate goals, and Set of final goals respectively.
$\mathcal{S}$	Rule-based system.
$\mathcal{R}, \mathcal{I}$, and $\mathcal{W}$	Rule base, Inference engine, and Working memory respectively.
i and f	Intermediate and final goal respectively.
g	A goal. The context clarifies whether g is intermediate or final.
h, k	Rule base hypotheses. The context clarifies if they are intermediate or final.
r and R	A rule in some rule base and a specific rule used in an example rule base respectively.
α and η	Accessibility set and enabling set (of a rule) respectively.
δ and Δ	Goal and rule base distance metrics respectively.
Φ and σ	Path and rule sequence (in a rule base) respectively.
Π (π)	Set (subset) of paths in a rule base.
ψ and ρ	Property and set of rules (in a rule base) respectively.
Θ	Rule base transformation.
$>$ and $\succ$	Rule attainability and accessibility respectively.
$\vdash$	$X \vdash Y$ should be read as "from X infer Y."
$\models$	$X \models Y$ should be read as "in model X properties Y hold."

Atoms Upper case letters, such as $A, B, C, \ldots$ denote atoms. The context clarifies whether an upper case letter is an atom or a predicate. Any word that is in upper case also denotes an atom.

Constants Lower case letters from the beginning of the alphabet such as $a, b, c, \ldots$, are used to denote constants. Any other word with its first letter capitalized is also a constant: for example, `Tom, Cathy,` ....

Variables Lower case letters from the end of the alphabet, such as $x, y, z, \ldots$, are used to denote variables. Any word that is in lowercase also denotes a variable.

In addition, if S is a set, then $|S|$ denotes the number of elements in the set S.
The definitions given in this thesis are either original, or considerably modified (if adopted from other sources) to correspond to the work of this thesis.

The notation that was used for the cost analysis of maintenance operations in chapter 3 (section 3.3.1) is summarized below.

Symbol	Represented Concept
C	Cost per unit of work (effort).
H_i, H_f	Set of intermediate and final hypotheses in a rule base respectively.
N_h	Number of rules associated with hypothesis h.
G_i, G_f	Set of intermediate and final goals respectively.
N_M, $N_{M'}$	Number of rules realizing M final goals and number of rules realizing M' intermediate goals respectively.
θ	Cost associated with checking system documentation.

A symbol may be augmented by an apostrophe, superscripts and/or subscripts whenever doing so improves the clarity of the presentation. Unless otherwise specified, the notation summarized above is followed in this thesis.

Bibliography

Aben, M. [1993]. "Formally Specifying Reusable Knowledge Model Components," *Knowledge Acquisition*, *5*(2), 119–141.

Agrawal, R., and Tanniru, M. [1992]. "A Petri-Net based Approach for Verifying the Integrity of Production Systems," *International Journal of Man-Machine Studies*, *36*(3), 447–468.

Andert Jr, E. P. [1993]. "Integrated Design and V&V of Knowledge-Based Systems," In *Notes of the Workshop on Validation and Verification of Knowledge-Based Systems (Eleventh National Conference on Artificial Intelligence)*, pp. 127–128 Washington D.C.

Antoniou, G., and Wachsmuth, I. [1994]. "Structuring and Modules for Knowledge Bases: Motivation for a New Model," *Knowledge-Based Systems*, *7*(1), 49–51.

Batarekh, A., Preece, A. D., Bennett, A., and Grogono, P. [1991]. "Specifying an Expert System," *Expert Systems with Applications*, *2*(4), 285–303.

Boehm, B. W. [1977]. "Seven Basic Principles of Software Engineering," In *Software Engineering Techniques (Infotech State-of-the-Art Report)*. Pergamon-Infotech, Maidenhead, UK.

Boehm, W. [1988]. "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, *21*(5), 61–72.

Bylander, T. C., and Mittal, S. [1986]. "CSRL: A Language for Classificatory Problem Solving," *AI Magazine*, *7*(3), 66–77.

Chander, P. G., Radhakrishnan, T., and Shinghal, R. [1995a]. "Design Schemes for Rule-based Systems," *Submitted to International Journal of Expert Systems: Research and Applications.*

Chander, P. G., Shinghal, R., and Radhakrishnan, T. [1995b]. "Goal Supported Knowledge Base Restructuring for Verification of Rule Bases," In *Notes of the Workshop on Verification & Validation of Knowledge-Based Systems (Fourteenth International Joint Conference on Artificial Intelligence)*, pp. 15–21 Montreal, Canada.

Chander, P. G., Shinghal, R., and Radhakrishnan, T. [1995c]. "Using Goals to Design and Verify Rule Bases," *Submitted to Decision Support Systems.*

Chander, P. G. [1995]. "Analyzing Properties of Design Schemes for Rule-based Systems," Computer Science Technical Report (August'95), Concordia University, Montreal, Canada.

Chander, P. G., Radhakrishnan, T., and Shinghal, R. [1995]. "Using Paths to Detect Redundancy in Rule Bases," In *Proceedings of the 11th IEEE Conference on Artificial Intelligence Applications (IEEE CAIA '95)*, pp. 133–139 Los Angeles, California.

Chander, P. G., Shinghal, R., and Radhakrishnan, T. [1994]. "Static Determination of Dynamic Functional Attributes in Rule-based Systems," In *Proceedings of the 1994 International Conference on Systems Research, Informatics and Cybernetics, AI Symposium (ICSRIC 94)*, pp. 79–84 Baden Baden, Germany.

Chandrasekaran, B. [1983]. "Towards a Taxonomy of Problem Solving Types," *AI Magazine*, *4*(1), 9–17.

Chandrasekaran, B. [1986]. "Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert System Design," *IEEE Expert*, *1*(3), 23–30.

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

Chen, Z., and Suen, C. [1993]. "Application of Metric Measures: from Conventional Software to Expert Systems," In *Notes of the Workshop on Verification and Validation of Knowledge-Based Systems (Eleventh National Conference on Artificial Intelligence)*, pp. 44–51 Washington D.C.

Clancey, W. [1983]. "The Advantages of Abstract Control Knowledge in Expert System Design," In *Proceedings of the Third National Conference on Artificial Intelligence (AAAI 83)*, pp. 74–78 Washington D.C.

Conte, C., Dunsmore, H., and Shen, V. [1990]. *Software Engineering Metrics and Models.* Benjamin Cummings, California.

Cragun, B. J., and Steudel, H. J. [1987]. "A Decision-table-based Processor for Checking Completeness and Consistency in Rule-based Expert Systems," *International Journal of Man-Machine Studies*, *26*(5), 633–648.

Debenham, J. [1992]. "Expert Systems Designed for Maintenance," *Expert Systems with Applications*, *5*(3), 233–244.

Forgy, C. [1981]. "OPS5 User's Manual," CMU technical report, Carnegie Mellon University.

Ghezzi, C., Jazayeri, M., and Mandrioli, D. [1991]. *Fundamentals of Software Engineering.* Prentice Hall, New York.

Giarratano, J., and Riley, G. [1993]. *Expert Systems: Principles & Programming (2nd edition).* PWS Publishing Company, Boston, MA.

Ginsberg, A. [1988]. "Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency & Redundancy," In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI 88)*, Vol. 2, pp. 585–589 St. Paul, Minnesota.

Giovanni, G. [1989]. "Assuring Adequacy of Expert Systems in Critical Application Domains: A Constructive Approach," In Hollnagel, E. (Ed.), *The Reliability of Expert Systems*, pp. 134–167. Halsted Press, New York.

Gokulchander, P., Preece, A., and Grossner, C. [1992]. "Path Hunter: A Tool for Finding the Paths in a Rule Based Expert System," DAI Technical Report DAI-0592-0012, Concordia University, Montreal Quebec.

Grossner, C., Gokulchander, P., Preece, A., and Radhakrishnan, T. [1993]. "Data Distribution in Organizations of Cooperating Expert Systems," In *Notes of the 12th International Workshop on Distributed Artificial Intelligence*, pp. 203–218 Hidden Valley, Pennsylvania.

Grossner, C., Lyons, J., and Radhakrishnan, T. [1991]. "Validation of an Expert System Intended for Research in Distributed Artificial Intelligence," In *Proceedings of the 2nd CLIPS Conference, Johnson Space Center*, pp. 365–381 Houston, Texas.

Grossner, C., Preece, A., Gokulchander, P., Radhakrishnan, T., and Newborn, M. [1994]. "Data Sharing Among Cooperating Rule-Based Systems," DAI Technical Report DAI-0394-0019, Concordia University, Montreal Quebec.

Grossner, C., Preece, A., Gokulchander, P., Radhakrishnan, T., and Suen, C. [1993]. "Exploring the Structure of Rule Based Systems," In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI 93)*, pp. 704–709 Washington D.C.

Guida, G., and Mauri, G. [1993]. "Evaluating Performance and Quality of Knowledge-Based Systems: Foundation and Methodology," *IEEE transactions in Knowledge and Data engineering*, *5*(2), 204–224.

Gupta, U. G. [1993]. "Validation and Verification of Knowledge-Based Systems: A Survey," *Journal of Applied Intelligence*, *3*(4), 343–363.

Hamilton, D., Kelley, K., and Culbert, C. [1991]. "State-of-the-Practice in Knowledge-based System Verification and Validation," *Expert Systems with Applications*, *3*(3), 403–410.

Jacob, R. J. K., and Froscher, J. N. [1990]. "A Software Engineering Methodology for Rule-Based Systems," *IEEE Transactions on Knowledge and Data Engineering, 2*(2), 173–189.

Kiper, J. D. [1992]. "Structural Testing of Rule-Based Expert Systems," *ACM Transactions on Software Engineering and Methodology, 1*(2), 168–187.

Krause, P., Fox, J., Neil, M. O., and Glowinski, A. [1993]. "Can We Formally Specify a Medical Decision Support System?," *IEEE Expert, 8*(3), 56-61.

Lee, S., and O'Keefe, R. M. [1993]. "Subsumption Anomalies in Hybrid Knowledge Bases," *International Journal of Expert Systems, 6*(3), 299–320.

Lee, S., and O'Keefe, R. M. [1994]. "Developing a Strategy for Expert System Verification and Validation," *IEEE Transactions on Systems, Man, and Cybernetics, 24*(4), 643–655.

Lieberman, H. [1995]. "Letizia: An Agent that Assists Web Browsing," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 95)*, Vol. 1, pp. 924-929 Montreal, Canada.

Liebowitz, J. [1989]. "Problem Selection for Expert Systems Development," In Liebowitz, J., and De Salvo, D. A. (Eds.), *Structuring Expert Systems: Domain, Design and Development*, pp. 3-23. Englewood Cliffs, New Jersey: Prentice Hall.

Liebowitz, J., and De Salvo, D. A. (Eds.). [1989]. *Structuring Expert Systems: Domain, Design and Development.* Englewood Cliffs, New Jersey: Prentice Hall.

Loiseau, S., and Rousset, M.-C. [1993]. "Formal Verification of Knowledge Bases Focused on Consistency: Two Experiments Based on ATMS Techniques," *International Journal of Expert Systems, 6*(3), 273-298.

Long, J. A., and Neale, I. M. [1993]. "Using Paper Models in Validation, Verification and Testing," *International Journal of Expert Systems, 6*(3), 357-382.

Lounis, H. [1993]. "Integrating machine-learning techniques in knowledge-based system verification," In *7th International Symposium in Artificial Intelligence*, pp. 405–414 Trondheim, Norway.

Lucas, P. [1994]. "Refinement of the HEPAR Expert System: Tools and Techniques," *Artificial Intelligence in Medicine*, *6*(2), 175–188.

Lunardhi, A. D., and Passino, K. M. [1991]. "Verification of Dynamic Properties of Rule-based Expert Systems," In *30th IEEE Conference on Decision and control*, pp. 1561–1566 Brighton, England.

Marcus, S., and McDermott, J. [1989]. "SALT: A Knowledge Acquisition Language for Propose-and-Revise Systems," *Artificial Intelligence*, *39*(1), 1-37.

McDuffie, R. S., Smith, L. M., and Flory, S. M. [1994]. "Validation of an Accounting Expert System for Business Combinations," *Expert Systems with Applications*, *7*(2), 175–183.

Mehrotra, M. [1993]. "Multi-Viewpoint Clustering Analysis," In *Notes of the Workshop on Verification and Validation of Knowledge-Based Systems (National Conference on Artificial Intelligence)*, pp. 49–56 Washington, D.C.

Mehrotra, M. [1995]. "Requirements and Capabilities of the Multi-Viewpoint Clustering Analysis Methodology," In *Notes of the Workshop on Verification and Validation of Knowledge-Based Systems (Fourteenth International Joint Conference on Artificial Intelligence)*, pp. 49-56 Montreal, Canada.

Mengshiel, O. J. [1993]. "Knowledge Validation: Principles and Practice," *IEEE Expert*, *8*(3), 62-68.

Meseguer, P. [1992]. "Incremental Verification of Rule-based Expert Systems," In Neumann, B. (Ed.), *10th European Conference on Artificial Intelligence*, pp. 829–834 Vienna, Austria.

Meseguer, P. [1993]. "Expert System Validation Through Knowledge Base Refinement," In *13th International Joint Conference on Artificial Intelligence (IJCAI'93)*, pp. 477-482 Savoie, France.

Mukherjee, R., and Gamble, R. [1995]. "Critical Examination of Subsumption Anomalies in Hybrid Systems," In *Notes of the Workshop on Verification and Validation of Knowledge-Based Systems (Fourteenth International Joint Conference on Artificial Intelligence)*, pp. 57–62 Montreal, Canada.

Nazareth, D. L. [1993]. " Investigating the applicability of Petri Nets for Rule-based System Verification," *IEEE Transactions on Knowledge and Data Engineering*, *4*(3), 447–468.

Nguyen, T. [1987]. "Verifying Consistency of Production Systems," In *Proceedings of the 3rd Conference on Artificial Intelligence Applications*, pp. 4–8 Washington, D.C.

Nguyen, T., Perkins, W., Laffey, T., and Pecora, D. [1985]. "Checking an Expert Systems Knowledge Base for Consistency and Completeness," In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 85)*, Vol. 1, pp. 375–278 Boston, MA.

O'Keefe, R. M., Balci, O., and Smith, E. P. [1987]. "Validating Expert System Performance," *IEEE Expert*, *2*(4), 81-90.

O'Keefe, R. M., and O'Leary, D. E. [1993]. "Expert System Verification and Validation: A Survey and Tutorial," *Artificial Intelligence Review*, *7*(1), 3–42.

O'Leary, D. E. [1995]. "Inference Engine Greediness and Subsumption of Conditions in Rule-based Systems," In *Notes of the Workshop on Verification and Validation of Knowledge-Based Systems (Fourteenth International Joint Conference on Artificial Intelligence)*, pp. 42–48 Montreal, Canada.

O'Neal, M. B., and Edwards Jr., W. R. [1993]. "Comprehending Rule-based Programs: A Graph Oriented Approach," *International Journal of Man-Machine Studies*, *39*(1), 147-175.

Plant, R. T. [1992]. "Expert System Development and Testing: A Knowledge Engineer's Perspective," *Journal of Systems Software*, *19*(2), 141–146.

Plant, R. T. [1993]. "The Meta Knowledge Level: A Methodology for Validation," In *Proceedings of the AAAI Workshop on Validation and Verification of Knowledge-Based Systems*, pp. 94–108 Washington D.C.

Polat, F., and Guvenir, H. A. [1993]. " UVT: A Unification-based Tool for Knowledge Base Verification," *IEEE Expert*, *8*(3), 69–75.

Pople, H. W. [1982]. "Heuristic Methods for Imposing Structure on Ill-Stru tured Problems," In Szolovits, P. (Ed.), *Artificial Intelligence in Medicine*, pp. 119–190. Westview Press.

Prakash, G. R., Subramanian, E., and Mahabala, H. [1991]. "A Methodology for Systematic Verification of OPS5-Based AI Applications," In *Twelfth International Joint Conference on Artificial Intelligence*, pp. 3–8 Sydney, Australia.

Preece, A., Gokulchandei, P., Grossner, C., and Radhakrishnan, T. [1993a]. "Modeling Rule Base Structure for Expert System Quality Assurance," In *Notes of the Workshop on Validation of Knowledge-Based Systems (Thirteenth International Joint Conference on Artificial Intelligence)*, pp. 37–50 Savoie, France.

Preece, A., Grossner, C., Gokulchander, P., and Radhakrishnan, T. [1993b]. "Structural Validation of Expert Systems: Exoerience Using a Formal Model," In *Notes of the Workshop on Validation and Verification of Knowledge-Based Systems (Eleventh National Conference on Artificial Intelligence)*, pp. 19–26 Washington D.C.

Preece, A., Grossner, C., Gokulchander, P., and Radhakrishnan, T. [1994]. "Structural Validation of Expert Systems: Experience Using a Formal Model," In Liebowitz, J. (Ed.), *Second World Congress on Expert Systems*, pp. 323–330 Estoril, Portugal.

Preece, A., Grossner, C., Gokulchander, P., and Radhakrishnan, T. [1995]. "Structure-Based Validation of Rule-Based Systems," *Submitted to Knowledge and Data Engineering.*

Preece, A. D. [1990]. "Towards a Methodology for Evaluating Expert Systems," *Expert Systems*, *7*(4), 215–223.

Preece, A. D. [1992]. "A Survey of Empirical Validation Techniques for Expert Systems," Report for Bell Canada, Centre for Pattern Recognition and Machine Intelligence, Concordia University, Montréal, Canada.

Preece, A. D. [1993]. "A New Approach to Detecting Missing Knowledge in Expert System Rule Bases," *International Journal of Man Machine Studies*, *38*(4), 661–688.

Preece, A. D., and Shinghal, R. [1992]. "Verifying and Testing Expert System Conceptual Models," In *IEEE International Conference on Systems, Man and Cybernetics*, pp. 922–927 Chicago, Illinois.

Preece, A. D., Shinghal, R., and Batarekh, A. [1992a]. "Principles and Practice in Verifying Rule-Based Systems," *Knowledge Engineering Review*, *7*(2), 115–141.

Preece, A. D., Shinghal, R., and Batarekh, A. [1992b]. "Verifying Expert Systems: A Logical Framework and a Practical Tool," *Expert Systems with Applications*. *3*(2/3), 421–436.

Rich, E. [1991]. *Artificial Intelligence (2nd edition)*. McGraw Hill, New York.

Rousset, M.-C. [1988]. "On the Consistency of Knowledge Bases: The COVADIS System," *Computational Intelligence*, *4*(2), 166–170. Also in *ECAI 88, Proc. European Conference on AI* (Munich, August 1-5, 1988), pages 79-84.

Rushby, J. [1988]. "Quality Measures and Assurance for AI Software," NASA Contractor Report CR-4187, SRI International, Menlo Park, California.

Schreiber, G., Wielinga, B., and de Hoog, R. [1994]. "CommonKADS: A Comprehensive Methodology for KBS Development ," *IEEE Expert*, *9*(6), 28-37.

Shinghal, R. [1992]. *Formal Concepts in Artificial Intelligence.* Chapman & Hall, London, U.K., co-published in U.S. with Van Nostrand, New York.

Simon, H. A. [1973]. "The Structure of Ill-Structured Problems," *Artificial Intelligence*, *4*, 181-201.

Suh, Y.-H., and Murray, T. J. [1994]. "A Tree-Based Approach for Verifying Completeness and Consistency in Rule-based Systems," *Expert Systems with Applications*, *7*(2), 199–220.

Suwa, M., Scott, A. C., and Shortliffe, E. H. [1982]. "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System," *AI Magazine*, *3*(4), 16–21.

Valiente, G. [1993]. "Verification of Knowledge Base Redundancy and Subsumption Using Graph Transformations," *International Journal of Expert Systems*, *6*(3), 341-355.

Vestli, M., Nordb∅, I., and S∅lvberg, A. [1994]. "Modeling Control in Rule-based Systems," *IEEE Software*, *11*(3).

Vignollet, L., and Lelouche, R. [1993]. "Test Case Generation Using KBS Strategy," In *13th International Joint Conference on Artificial Intelligence (IJCAI 93)*, pp. 483–488 Savoie, France.

Vinze, A. S. [1992]. "Empirical Verification of Effectiveness for a Knowledge-based System," *International Journal of Man-machine Studies*, *37*(3), 309–334.

Waldinger, R. J., and Stickel, M. E. [1991]. "Proving Properties of Rule-based Systems," In *Proceedings of the IEEE 7th Conference on AI Applications*, pp. 81–88 Miami Beach, Florida.

Weitzel, J. R., and Kershberg, L. [1989]. "Developing Knowledge-Based Systems: Reorganizing the System Development Cycle," *Communication of the ACM*, *32*(4), 482–488.

Wells, S. A. [1993]. "The VIVA Method: A Life-Cycle Independent Approach to KBS Validation," In *Proceedings of the AAAI Workshop on Validation and Verification of Knowledge-Based Systems*, pp. 109–113 Washington D.C.

Yen, J., Juang, H.-L., and MacGregor, R. [1991]. "Using Polymorphism to Improve Expert System Maintainability," *IEEE Expert*, *6*(2), 48–55.

Yost, G. R. [1993]. "Acquiring Knowledge in SOAR," *IEEE Expert*, *8*(3), 26–34.

Yost, G. R., and Newell, A. [1989]. "A Problem Space Approach to Expert System Specification," In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI 89)*, pp. 621–627 San Mateo, California.

Zlatareva, N., and Preece, A. D. [1994]. "State of the Art in Automated Validation of Knowledge-Based Systems," *Expert Systems with Applications*, *7*(2), 151–167.

Zualkernan, I. A., and Lin, Y.-J. [1993]. "Experimental Evaluation of Output Based Partition Testing for Expert Systems," In *Proceedings of the 1993 IEEE International Conference on Tools for AI*, pp. 190–197 Boston, Massachusetts.