Spring 1991

Effectiveness Analysis of Knowledge Bases

Shensheng Zhao
Old Dominion University

EFFECTIVENESS ANALYSIS OF KNOWLEDGE BASES

by

Shensheng Zhao

B.S. September 1969, Beijing University, P.R.C
M.S. August 1986, Old Dominion University, U.S.A

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy

Computer Science

Old Dominion University
January, 1991

Approved by:

____
Stewart N.T. Shen (Director)

____

____

____

____

ABSTRACT

EFFECTIVENESS ANALYSIS OF KNOWLEDGE BASE

Shensheng Zhao
Old Dominion University, 1990
Director: Dr. Stewart N.T. Shen

Knowledge base systems (expert systems) are entering a critical stage as interest spreads from university research to practical applications. If knowledge base systems are to withstand this transition, special attention must be paid to checking their effectiveness. The issue of effectiveness analysis of knowledge base systems has been largely ignored and few works have been published in this field. This dissertation shows how the effectiveness of a knowledge base system can be defined, discussed and analyzed at the knowledge base system level and the knowledge base level. We characterize the effectiveness of a knowledge base system in terms of minimality, termination, completeness and consistency. In order to resolve these issues, we propose a general framework for checking these properties. This framework includes models **KBS** and **KB** for knowledge base systems and knowledge bases, respectively. These models provide an environment in which we can discuss and analyze the effectiveness of a knowledge base system. This framework leads to analysis for rule set properties and a set of problem formulations for each of the following: minimality, termination, completeness and consistency. In this dissertation, we have designed a set of algorithms for resolving these problems and given some computational results.

Table of Contents

ACKNOWLEDGEMENTS

There are several people I would like to thank.

First, I wish to thank my advisor, Dr. Stewart N. T. Shen, for his guidance and enthusiastic support throughout my graduate education, dissertation and preparation for a successful future of continued work in computer science.

I also would like to thank the other members of my committee, Dr. Larry W. Wilson, Dr. Christian Wild, Dr. Nageswara S. V. Rao and Dr. Meng-Sang Chew, for their time and helpful advice in reviewing this dissertation.

Without the love and support of my wife, Jingying Zhang, this dissertation would not have been written. Thanks also to my parents, Hongsen Zhao and Zunqing Yang.

Thanks to Old Dominion University faculty members: Dr. Michael Overstreet, Dr. James L. Schwing, Steve Daniel and Dennis E. Ray for their help in the preparation of this dissertation.

To my fellow graduate students, past and present, at Old Dominion University, Ghassan Issa, Jih-shih Hsu, Myron Xu and Susan Schwartz, thank you for your friendship and help.

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Knowledge base systems are products of social cooperative activities and constitute a discipline that lies on the boundary of several fields. Knowledge base systems (or expert systems) store and employ human knowledge to solve problems that ordinarily require human intelligence. Numerous knowledge base systems, developed or being developed, have been reported in literature.

The importance of knowledge base systems (or expert systems) has been growing in industrial, medical, scientific, educational and other fields during the past two decades. Several major reasons for this are: (1) the necessity of handling an overwhelming amount of knowledge in these areas; (2) the potential of knowledge base systems to solve certain problems and to train new experts; (3) the easy implementation by using expert system shell; (4) the desire to capture corporate knowledge so it is not lost as personnel change [89]. In medicine, for example, a knowledge base system named HELP is used at LDS Hospital in Salt Lake City, Utah, where it has proven effective in reducing health care costs [63]. HELP analyzes patient data when a test order or result is entered into the system, and it warns physicians about such things as drug-drug interactions and drug contra-indications. Studies showed that 80% of HELP's drug-drug interaction alerts were used by physicians to change prescriptions, test orders, or other forms of treatment, all of which resulted in shortened hospital stays and improved quality of care. Examples of knowledge base systems developed for industrial tasks include PROSPECTOR, which successfully has been used to locate mineral deposits [21], and R1 (XCON), which is used to construct computers [47]. In the educational area, GRANT finds sources of funding for research proposals [15]. GRANT finds funding agencies based on semantic matches between research proposals and the agencies' research interests. For example, if a research proposal mentions hemoglobin, GRANT will find agencies that support research on blood, even if they do not specifically mention hemoglobin in their statements of interest. GRANT "thinks" that hemoglobin is a component of blood. Today's knowledge base systems solve relatively simple problems, give some explanations or make some construction [90]. The Gartner Study, a report released in June 1988, reported that the number of deployed expert systems increased from 50 in 1987 to 1,400 in 1988, and the number of expert systems under development increased from 2,500 to 8,500 during the same period [65].

Even though knowledge base systems (or expert systems) are among the most active applications from artificial intelligence research efforts, the issue of effectiveness analysis has been largely ignored [9,16,26,33]. The lack of effectiveness analysis for knowledge base systems could lead to personal and financial disaster if an erroneous or inappropriate knowledge base system is used in the real world. However, there are very few published reports which describe analysis for such systems. Exceptions include the validation of MYCIN's performance [93,94], and evaluating reports for PROSPECT and INTERNIST-1 [28,48]. The growing reliance on knowledge base systems in real applications requires the development of a theoretical basis and methods to check such systems in order to ensure their effectiveness.

1.1. Effectiveness of Knowledge Base System

There are many dimensions by which we might observe and judge a knowledge base system. These make the effectiveness of knowledge base systems a very difficult problem to define and deal with. To date, little work has been done in this area [8, 12, 16, 38, 55, 64, 72].

Gaschnig, Klahr, Pople, Shortliffe and Terry [29] point out that knowledge base systems are unique because they contain human expertise. Since, no one knows how to evaluate human expertise adequately, this leads to difficulty in comparing and evaluating a knowledge base with respect to human performance. Hayes-Roth [35] describes several good features of rule-based knowledge base systems and points out that one of the key features these systems lack is "a suitable verification methodology or a technique for testing the consistency and completeness of a rule set." Green [34], Geissman and Schultz [74] emphasize that formal verification and validation are necessary for acceptance of knowledge base system into critical areas. They also point out that vague objectives for knowledge base design make it hard to find testable requirements. This leads into a vicious circle: "Since we don't know how to check the knowledge base so we should check it" [34]. All researchers in this field have indicated that verifying and validating the knowledge base is a very significant and important problem, but few works have been reported. Efforts have been made to break the vicious circle. Some researchers have worked on this problem, however these works are primitive and leave room for much improvement. The perfect methodology or technique for analyzing the effectiveness of a knowledge base does not exist. This dissertation improves upon previous efforts.

There are many different kinds of knowledge base systems. They can be classified according to the method of the knowledge representation and the inference mechanisms. Many knowledge base systems are still in the research phase, such as nonmonotonic reasoning systems [43]. In this dissertation, we propose a method for checking a restricted class of rule-based systems which are monotonic, non-recursive and propositional with no certainty factors. The models and mechanism used for such systems will be introduced later (see Chapter 4).

In this dissertation we say that a knowledge base system is effective if it possesses the following properties:

- Minimality
- Termination
- Completeness
- Consistency.

By way of introduction, these properties are briefly defined as follows (detailed definitions will be given in later chapters):

Minimality: The knowledge base system does not contain any redundant rules, which are classified as *self redundant rule, redundant rule* and *redundant triangle.* (see Chapter 6)

Termination: Considering a knowledge base as a network, we identify three kinds of cycles: *strong apple cycle, semi strong apple cycle* and *weak apple cycle.* The sufficient and necessary condition for termination of a knowledge base is that it does not contain any firable cycles. (see Chapter 7)

Completeness: For a particular knowledge domain, the knowledge base should contain all the necessary information about objects and the relationships among these objects. (see Chapter 8)

Consistency: A knowledge base is consistent if it is evidence consistent, hypothesis consistent and rule consistent. (see Chapter 9)

In order to discuss these issues, we propose a general framework which contains models **KBS** and **KB** for knowledge base systems and for knowledge bases, respectively. These models are domain independent and rule-based. Based on these models, we can analyze the rule properties and investigate the properties of minimality, termination, completeness and consistency. Some sample systems are also analyzed.

Considering the definition of knowledge base system effectiveness, there appear to be two quite different categories in the literature. In the first category, researchers discuss sufficient conditions for the effectiveness of knowledge base system. For example, Marcot [46] proposed the following criteria : *accuracy, adaptability, adequacy, appeal, availability, breadth, depth, face validity, generality, precision, realism, reliability, resolution, robustness, sensitivity, technical and operational validity, Turing test, usefulness, plain, validity* and *wholeness*. Another proposed criterion is that a knowledge base system should "act like an expert" and demonstrate deep knowledge [45,87]. The definitions for certain terms in these criteria should be defined precisely. These sufficient conditions for knowledge base effectiveness are too strong to realize.

In the second category, researchers focus on some necessary conditions for the effectiveness of knowledge base systems. They check the completeness and/or consistency of a knowledge base system. Checking these necessary conditions for effectiveness is a realistical job. The definition for effectiveness used in this dissertation belongs to the second category. We will briefly review previous works in Chapter 3. There are three major differences between our definition of effectiveness and definitions stated in previous works: (1) ours is more systematically defined; (2) it contains termination property, which is ignored in most previous works; (3) it is based on knowledge base models (see Chapter 4).

In previous decades, quite a few works were done in this field. In this dissertation, we do not attempt to make a knowledge base system fit the sufficient conditions of its effectiveness. We propose an elementary framework for checking some necessary conditions of the effectiveness of knowledge base systems according to our definition.

1.2. Contributions of This Dissertation

The main contributions of this dissertation are:

(1) a model-based methodology for analyzing the effectiveness of knowledge base systems; and

(2) computational results which check the effectiveness of knowledge base system.

1.2.1. Models for Knowledge Base System and Knowledge Base

The methodological contributions of this dissertation are in the form of two models: one for knowledge base systems and the other for knowledge bases. Based on such models, we can define the effectiveness problem of knowledge base systems.

The effectiveness analysis or performance evaluation has long been a neglected concern for knowledge base systems. Major reasons for this neglect are:

(1) Most knowledge base system projects have been research or proof-of-principle oriented. That is, the major concern has been whether a computer system could possibly be constructed to do a given task. The questions of how well the system performed relative to a human expert or some other performance standard and the cost of the system were usually of secondary interest and often ignored.

(2) Criteria for measuring the effectiveness of a knowledge base system are very hard to define. These criteria can be defined only after considering many factors. Even the same criteria can be interpreted with different meanings for different systems and for different stages of the life cycle in one system.

(3) There have been no general models available which could be used to analyze the effectiveness of knowledge base systems. Since knowledge base systems are task or domain problem oriented, this hampers the construction of general models.

We propose a general framework which contains two models, namely the **KBS** model for knowledge base systems and the **KB** model for knowledge bases. In the **KBS** model, we look at a knowledge base system as a product of cooperative and interactive activities, resulting from interdisciplinary research. This model is concerned with the knowledge base system as a whole, together with its environment, such as domain characteristics, problem features, domain experts, users and AI technology issues. The primary purpose of effectiveness analysis based on the **KBS** model is to obtain an overview, or a general outline of the significance, performance, functionality, achievements and limitations of the system.

We define the **KBS** model of knowledge base systems as a five tuple,

$$\mathbf{KBS} = (\ \mathbf{U}, \mathbf{T}, \mathbf{R}, \mathbf{F}, \mathbf{M}\ ).$$

- **U**: application domain universe
- **T**: relational table
- **R**: set of rules
- **F**: set of functional specifications
- **M**: mechanism for the knowledge base system

The **U** is *an application domain universe.* The application domain universe contains finite knowledge elements which can be notated as

$$\mathbf{U} = \{\ u_1, u_2, \ldots, u_n\ \}.$$

The **T** is a *relational table.* It is articulated knowledge from domain experts, and it records and describes the expertise (knowhow) of experts in an application domain. The relational table **T** contains finite tuples which can be notated as:

$$\mathbf{T} = \{\ t_1, t_2, \ldots, t_m\ \}.$$

The **R** is *a set of rules.* The rules are the encoded expertise of experts about certain application domain. The set of rules **R** can be notated as

$$\mathbf{R} = \{\ r_1, r_2, \ldots, r_k\ \}.$$

The **F** is *a set of functional specifications.* The functional specifications are derived from project objectives, user requirements and design specifications. The **M** is *a mechanism.* The mechanism **M** includes the processor (inference engine) and interface.

On the other hand, the **KB** model is concerned with specific, detailed internal structural properties and behaviors of knowledge base systems. When operating at this level of analysis, we put a knowledge base system, or some parts of the knowledge base system, under close study to observe the details of its effectiveness. In particular, we analyze and check the knowledge base, which is the kernel component of the knowledge base system. In order to achieve this goal we view a knowledge base as a formal system.

We define the **KB** model for knowledge base, which is adopted from Reiter [69]. A knowledge base is a triple:

KB = (E, H, R), where

(1) **E** : a finite nonempty set of evidence $\{ e_1, e_2, ..., e_n \}$

E represents all observable facts, symptoms, conditions, or askable findings;

(2) **H** : a finite nonempty set of hypotheses $\{ h_1, h_2, ..., h_m \}$

H represents all conclusions, actions, diagnostics or explanations;

(3) **R** : a finite nonempty set of IF - THEN rules, $\{ r_1, r_2, ..., r_k \}$.

R represents all associative knowledge, or encoded expertise.

The rules are special mapping functions, there are three types of rules:

Type I : From Evidences to Hypotheses (**E** $\rightarrow$ **H**), notated as $\mathbf{R_I}$.

Type II : From Evidences $\times$ Hypotheses to Hypotheses (**E** $\times$ **H** $\rightarrow$ **H**), notated as $\mathbf{R_{II}}$.

Type III : From Hypotheses to Hypotheses (**H** $\rightarrow$ **H**), notated as $\mathbf{R_{III}}$.

Based on the **KBS** and **KB** models, we examined and checked the minimality, termination, completeness and consistency problems.

The goal of introducing these models is to provide a well defined problem setting for the analysis of knowledge base systems and knowledge bases.

1.2.2. Results Concerning the Complexity of Checking KBS(KB) Effectiveness

Effectiveness checking of **KBS(KB)** implies finding deficiencies in the system with respect to minimality, termination, completeness and consistency. Systematical analysis of the complexity of checking **KBS(KB)** effectiveness has not been carried out. Based on the **KBS** and **KB** models, our main results about the complexity of checking **KBS(KB)** effectiveness are summarized as follows :

(1) At the **KBS** level:

The minimality problem, termination problem and consistency problem are very hard to define at the **KBS** level. For the completeness problem of **KBS**, we have identified *functional completeness* and *relational completeness*. The functional completeness of a **KBS** requires that the **KBS** satisfy functional specifications, and the relational completeness requires that the rule set of the **KBS** exactly cover the relational table of the **KBS**. The functional completeness of the **KBS** is provable or testable for restricted domain. The Minimal Exact Rule Cover Problem (**MERC** problem) of the **KBS** belongs to NP-Complete.

(2) At the **KB** level:

Based on the model **KB**, we have formulated the minimality problem, termination problem, completeness problem and consistency problem. A **KB** is minimal if it does not contain any self redundant, redundant, subsumed redundant and redundant triangle rules (see Chapter 6). A **KB** is terminated if it does not contain any strong, semi strong, and weak cycles (see Chapter 7 for detail). A **KB** is rule complete (**r**_complete) if all the evidences are defined and all hypothesis are satisfiable or reachable (see Chapter 8). A **KB** is consistent if it satisfies the conditions for evidential, hypothetical and rule consistency (see Chapter 9). According to these definitions, we have designed a set of algorithms for solving such problems and also presented some computational results. Before we present our computational results, let us introduce some notations and assumptions:

$|\mathbf{T}| = T$: The total number of tuples in relational table.

$|\mathbf{R}| = K$: The cardinality of rule set **R**.

$|\mathbf{R_I}| = K_1$: The cardinality of rule set $\mathbf{R_I}$.

$|\mathbf{R}_{\mathrm{II}}| = K_2$: The cardinality of rule set $\mathbf{R}_{\mathrm{II}}$.

$|\mathbf{R}_{\mathrm{III}}| = K_3$: The cardinality of rule set $\mathbf{R}_{\mathrm{III}}$.

$|\mathbf{E}| = N$: The cardinality of evidence set **E**.

$|\mathbf{H}| = M$: The cardinality of hypotheses set **H**.

L : The average size of rule r, r $\in$ **R**, the average number of knowledge elements in the rule r.

L_e : The average number of evidences in the rule r, r $\in$ **R**.

L_h : The average number of hypotheses in the rule r, r $\in$ **R**.

LHS(r) (or RHS(r)): The set of knowledge elements in left (right) hand side of rule r.

q : The number of groups of rules, which are partitions according to the size of LHS(r), r $\in$ **R**.

We assume the existence of the following operations:

- membership checking, e.g., e $\in$ LHS(r), r $\in$ **R**.
- intersection operation, e.g., LHS(r_i) $\cap$ RHS(r_j), $r_i, r_j \in$ **R**
- subset checking, e.g., LHS(r_i) $\subseteq$ RHS(r_j), $r_i, r_j \in$ **R**.
- instance checking, e.g., a tuple t_i is an instance of rule r_j or is covered by rule r_j.

Also we assume that the costs of the operations above are constant or polynomial time. If we store the information of all evidences **E** and all hypotheses **H** in two arrays respectively, these operations (membership, intersection and subset checking) can be efficiently executed. We assume the existence of a mechanism which can be used to check instances quickly. The development of that mechanism is of research interest in itself. In the following discussion, we will use the term *tractable* to refer that the order of computation is polynomial and *intractable* to refer that the order of computation is exponential. Also we will assume that the operations above use unit time. We desplay our results in Table 1.1 and compare these results with previous works in Table 1.1.

Model	Problem	Complexity
KBS	Minimality	Hard to define
KBS	Termination	Hard to define
KBS	Functional Completeness	Restrictedly Provable/Testable
	Relational Completeness	O($T^2 \times L^3$)
	Minimal Exact Cover Rule Set	NP-Complete
KBS	Consistency	Hard to define
KB	Minimality	
	Self Redundant Rules	O($K \times L$)
	Redundant Rules	O($½K^2/q$)
	Subsumed Rules	O($½K^2/q$)
	Redundant Triangle	O(K_3^3)
KB	Termination	
	Strong Cycle Rules	O(K_3^3)
	Semi Strong Cycle Rules	O($K_3 \times L$)
	Weak Cycle Rules	O($(K_2 + K_3) \times L$)
KB	Rule Completeness	O($N + M + K$)
KB	Consistency	
	Evidence Consistency	O($K_2 \times L^2$)
	Hypothesis Consistency	O($K \times L^2$)
	Rule Consistency Directly Conflicting	O($½K^2$)
	Rule Consistency Potentially Conflicting	O($K_1^2 + K_1 \times K_3$)

Table 1.1 Summary of the computational results of checking **KBS** and **KB**

Model	Problem	Suwa	Nguyen	Cragun	Ginsberg	Zhao
KBS	Functional Completeness	No	No	No	No	Yes
	Relational Completeness	Yes	Yes	Yes	No	Yes
	Minimal Exact Cover Rule Set	No	No	No	No	Yes
KB	Minimality					
	Self Redundant Rules	No	No	No	No	Yes
	Redundant Rules	Yes	Yes	Yes	Yes	Yes
	Subsumed Rules	Yes	Yes	Yes	Yes	Yes
	Redundant Triangle	No	No	No	No	Yes
KB	Termination					
	Strong Cycle Rules	No	No	No	No	Yes
	Semi Strong Cycle Rules	No	No	No	No	Yes
	Weak Cycle Rules	No	No	No	No	Yes
KB	Rule Completeness	No	Yes	Yes	Yes	Yes
KB	Consistency					
	Evidence Consistency	No	No	No	No	Yes
	Hypothesis Consistency	No	No	No	No	Yes
	Rule Consistency Directly Conflicting	Yes	Yes	No	Yes	Yes
	Rule Consistency Potentially Conflicting	No	No	No	Yes	Yes

Table 1.2 Comparisons with other works

1.3. Significance of This Dissertation

The fundamental importance of knowledge base systems is their potential for many applications. The major stumbling block preventing further use of knowledge base systems is a lack of methodology or technique for their effectiveness analysis. We illustrate the problem significance from operational and theoretical views.

From an operational point of view, knowledge base systems will not be used for critical applications (such as manned space missions, medical diagnosis, on on the battle field) until we can prove that they are free of catastrophic errors. Effectiveness analysis can guide the development of a knowledge base system, find the bugs in the system during the early phase of system development and improve the quality of the knowledge base system. This problem has attracted more and more attention from researchers and companies. At the ALVEY Conference (1987), many researchers and companies emphasized that the evaluation of knowledge base systems is an urgent task for further development and application [1]. In 1987, NASA held a special workshop on knowledge base system verification, at NASA/Ames Research Center, Mountain View, Calif. In March 1990, the TEST TECHNOLOGY SYMPOSIUM was sponsored by Advanced Test Technology Concepts Division U.S. Army Test and Evaluation Command at the Johns Hopkins University. This symposium concentrated on artificial intelligence/expert systems and robotics development of methodologies, test beds and software relating to the testing of systems incorporating AI technology.

From a theoretical point of view, effectiveness analysis of knowledge base systems is a new field. There has been much interest recently in this problem, and many elements of test, verification and validation exist in current knowledge base system methodologies; however, because of the infancy of knowledge base technology, these elements have not yet been assembled and standardized. There is a need for theoretical, systematic consideration and formal treatment for the knowledge base effectiveness analysis problem. This situation was challenging in that we faced a problem which is different from "traditional" ones (such as algorithm analysis, operating system evaluation, or software engineering).

This situation also gave us the opportunity to explore and develop new concepts, models and techniques for effectiveness analysis of knowledge base systems.

Karp [26,39] said,

> "I realize that certain areas in computer science have to be dominated by empirical investigations, but that doesn't relieve us of the responsibility of thinking very hard about what it is we're measuring, what it is we're trying to achieve, and when we can say that our design is a success. And I believe that a certain measure of scientific method is called for. I don't buy the idea that simply because you're simulating the somewhat unknowable cognitive processes of humans you are relieved of the obligation to have precise formulations of what you're doing."

Yes, we do have the obligation to think very hard and try to formulate precisely what we are doing in the field of knowledge base systems.

1.4. Organization of This Dissertation

Chapter 2 discusses some background issues: we compare the knowledge base system development life cycle with the software development life cycle, and compare the characteristics of knowledge base system analysis with traditional algorithm analysis. Our work is a part of knowledge base system analysis. Chapter 3 surveys the related work in effectiveness analysis of knowledge base in past decades and summarizes our research direction. Chapter 4 introduces our model **KBS** for knowledge base systems and model **KB** for knowledge bases. We also introduce the tool Knowledge Base Net **(KBNet)** which is used later for analyzing some properties of rule set, minimality and termination problems. Chapter 5 explores the properties of the rule set of a knowledge base and introduces Rule Dependency Graph (**RDG**). We formulate and discuss the problems for minimality, termination, completeness and consistency of knowledge bases in the remaining chapters. These four problems comprise the basis for the effectiveness of knowledge bases as we defined. Chapter 6 discusses the minimality problem. Chapter 7 is concerned with the termination problem. Chapter 8 focuses on the completeness problem. Chapter 9 discusses the consistency problem. In each of these chapters, we provide some algorithms for checking corresponding issues and also give some computational results. Chapter 10 gives a summary and conclusions, followed by a brief discussion of future research directions.

2.

CHAPTER 2

BACKGROUND

In the previous chapter we said that a knowledge base system is effective if it satisfies the following properties: minimality, termination, completeness and consistency. In this chapter we discuss some background issues as we compare the knowledge base system development life cycle with the software development life cycle and the characteristics of knowledge base system analysis with traditional algorithm analysis. Knowledge base system analysis is related to the **KBS** life cycle and affected by many factors, namely, model, human beings, domain, implementation, criteria and methods. To avoid confusion, here we use the term *knowledge base system analysis* (or **KBS** analysis) instead of effectiveness analysis. The effectiveness analysis of a **KBS** is a part of **KBS** analysis.

2.1. Knowledge Base System Development vs Software Development

We can see that knowledge base implementation is a software development process. They have much in common: The two are more alike than they are different [71]. Therefore, we can take advantage of the effort and methodologies that have been developed in the study of software engineering. However, there are significant differences between the traditional software development process and that used for knowledge base systems.

2.1.1. Software Development Life Cycle

The formulation of software engineering concepts started with the identification of the following two basic problems:

(1) programming was considered a black or semi-black art;

(2) complex programs were very hard to understand.

In order to solve these problems, software engineering focused on developing methodologies and tools to make programming activities more scientific. A formal methodology is intended to provide visibility to the process and to force discipline into it. The hope is to involve more people in the activity and manage the process by using formal methodology to develop a plan and check ongoing status against the plan. Tools were developed to assist the developer in managing the complexity and performing routine tasks. The key concept of software engineering is a software life-cycle model. A simple life-cycle model is shown in Figure. 2.1:

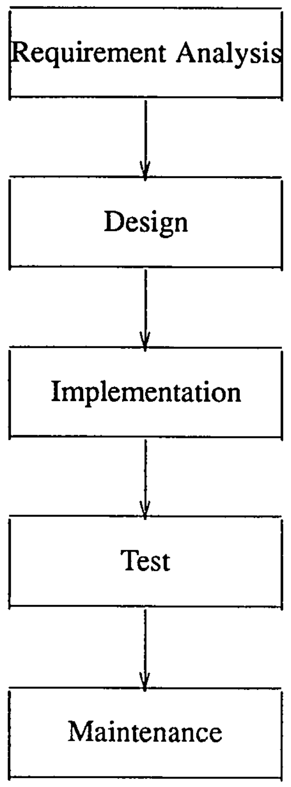

Figure 2.1 Software development life cycle model

This model defines the basic phases of a software development project and describes the output of each phase. The life cycle describes the process as follows:

It begins with the expression of user needs. A system analyst works with the user to analyze the task and define the functional requirements of the proposed systems. These requirements are formally described in a requirements specification. Following a review and approval of the specification, a design, which is described in a design specification, is developed to meet the requirements. Testing strategies also are outlined at this point. The design is implemented by using a representation (a particular programming language). It is then verified through testing. After the test phase, the system is released to the customer and a long period of maintenance begins.

The use of the life-cycle model is supposed to reduce the software crisis, but there are still several fundamental problems. Problem 1: The traditional life-cycle model is *linear*. "linear" means that each step in the model could be completely and correctly implemented before moving onto the next, which prevents *parallel* activities. Problem 2: The traditional life-cycle model omits the *iterative complexity* in the software development process. The "iterative complexity" means that iteration is inevitable in any large software development project. The software development life cycle in Figure 2.2 is closer to real activity :

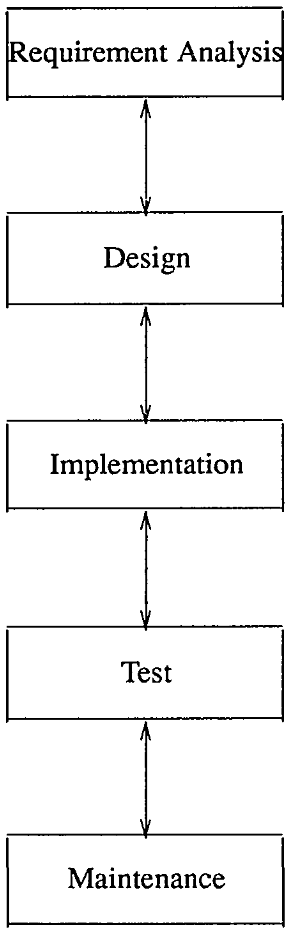

Figure 2.2. Iterative software development life cycle model

2.1.2. Knowledge Base System Development Life Cycle

Knowledge base system development generally follows a model (shown in Figure 2.3) that is similar to the iterative version of the software development life cycle model. The knowledge base system life-cycle model is based on a recognition of the evolutionary nature of software development. That is especially true of knowledge base system development because knowledge base systems deal with problem areas that initially are relatively poorly defined and not well understood.

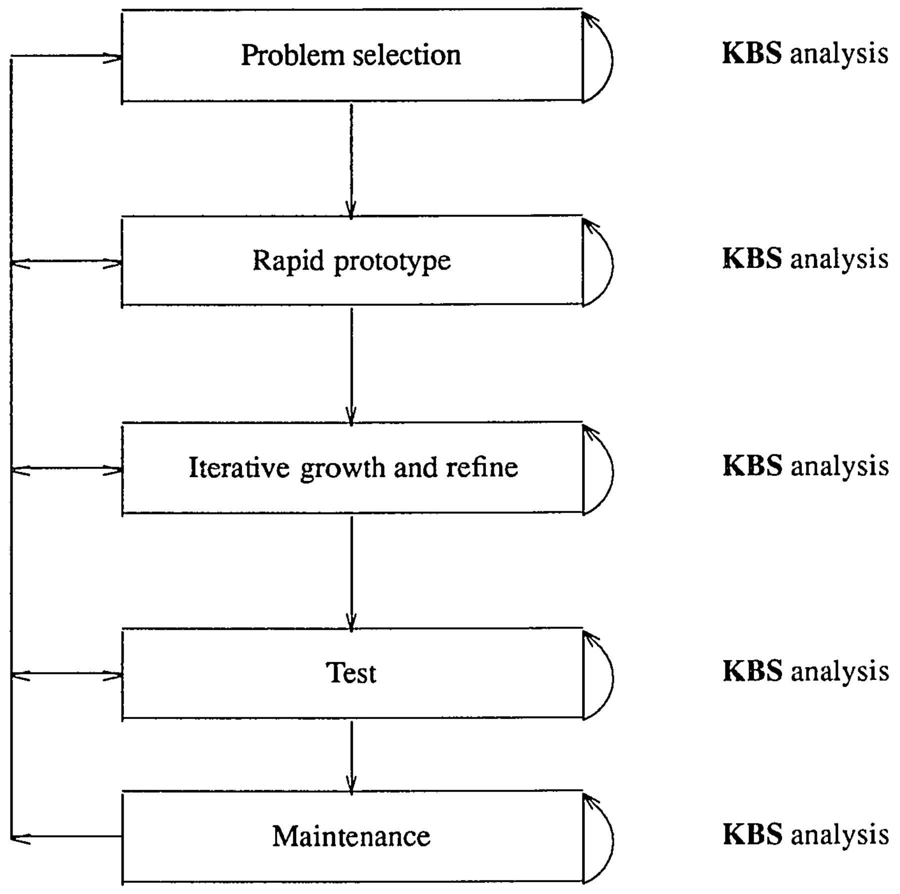

Figure 2.3 Iterative **KBS** development life cycle model

Stage 1: Problem selection.

The first step in a knowledge base system development project corresponds to the requirements analysis stage in traditional software development. The major objective is to ensure that the project will satisfy a real need and be technically feasible. The major work is to survey problems, select candidate application and task, and consider existing knowledge base technology (expert shell, algorithm, or creative view and ideas). From the effectiveness analysis viewpoint, this stage is the most important step for both research and application. This stage decides the direction for developing future knowledge base system. This stage produces the statement of potential knowledge base system, functions, methods and knowledge base requirements. This forms the basis of a requirements document. The requirements are derived from hign level effectiveness analysis, even though they may not be very detailed at this point, but they will help focus later development and its corresponding **KBS** analysis.

Stage 2: Rapid prototype.

The major objective of the rapid prototype stage is to quickly demonstrate the technical and economic feasibility of the intended knowledge base system. Domain experts and users evaluate an early prototype as a way of stating system requirement relatively precisely and providing a testbed for knowledge base development. The initial prototype is typically concerned with only a subset of the problem and may not have a sophisticated interface. In this stage, the **KBS** analysis of this prototype will give preliminary analysis of the knowledge base system, both positive and negative. Based on the prototype, more detailed requirements and specifications will be derived.

Stage 3: Iterative growth and refinement.

Following the initial prototype and project approval, the objective of the iterative growth and refinement is to develop the range of functions needed to deal with the full complexity of the specific problem. At this stage, the improvements include expansion, rewriting and checking of the knowledge base system. The iterative growth and refinement phase produces the more detailed statements of requirements and functional specifications.

Stage 4: Test.

Once the knowledge base system is performing well in most of the cases with which it is developed, it is appropriate to turn to a more structured **KBS** analysis of its performance. The principal objective at this stage is to show that the system is acceptable to the users for whom it is intended. From the literature, only a few knowledge base systems have reached this stage of **KBS** analysis. The principal examples are the PROSPECTOR [28], MYCIN [93,94], R1 (XCON) [47] and INTERNIST-1 [48]. The positive results from this stage will convince the end users of the competence of the system, any negative results will guide new research activity or improvements, and the system development will return to previous stages for additional refinement.

Stage 5: Maintenance.

At this stage we should have firm plans for maintaining the knowledge base system, monitoring its performance and finding problems (bugs). The system's credibility is largely dependent on previous stages. The solid performance data will support claims about the quality of the system's performance. The deficiencies of the system's performance will suggest further refinements and modifications.

Between October 1987 and June 1988 O'Neill and Morris [56] sent a short questionnaire to 52 software houses specializing in knowledge base work, asking for information on tools, projects and staff. They reported some results from their survey about the current state of the knowledge base development methodologies. Table 2.1 below shows that the rule-based paradigm still appears to be the most prominent form of knowledge presentation, although more organizations are starting to use other techniques.

Form of representation	%
Rules	56
Everything	17
Frames	10
Semantic nets	7
Decision trees	5
Object-oriented programming	5
	100

Table 2.1 Forms of representation used

Table 2.2 indicates the nature and degree of user consultation, interactive process of the knowledge base system development cycle.

When users consulted	%
At outset and throughout project	35
During and after system development	30
Varies according to project	23
Did not consult users	12
	100

Table 2.2 User consultation

Table 2.3 shows that most companies used some form of human expertise in the development of their knowledge base systems, although it varied according to the nature of the problem to be solved. In many technical areas, expertise was readily available in the form of rule books and manuals, and some systems were designed to replace inadequate human performance or to co-exist with process control procedures using no human input.

Number of experts consulted	%
One	59
Multiple	22
Varied	15
None	4
	100

Table 2.3 Consultation with experts

These tables give us some solid data about characteristics of knowledge base systems. To summarize our discussion, we point out several unique characteristics of the knowledge base system development cycle:

- The domain experts and users are involved throughout the entire process. This is much different from the traditional software development, in which the users specify the requirements and then walk away and wait for the final products.
- Change and modification at each stage are viewed as normal. Specifically, change is strongly encouraged during the problem selection and prototype stages [38,79]. Feedback from the user and domain expert is the main source for making system improvements.
- Frequent demonstrations of work at each stage of development are encouraged. These demonstrations allow the user and domain expert to check and visualize the functionality of the knowledge base system and to propose or request changes.
- The knowledge base system is truly a product of social cooperative activities and a discipline that lies on the boundary of several fields: particular technical domain, computer science and human-machine interface.

- The production rule, or rule-based paradigm, still appears to be the most important form of knowledge representation for developing knowledge base systems or expert systems.

The traditional life-cycle model of software development and maintenance presumes that problems are relatively well specified. An alternative model for developing knowledge base systems is an exploratory model, in which problem definition, knowledge representation, problem solution and methods employed are mutually reinforcing. The process of developing knowledge base systems is a very complicated, highly iterative and evolutionary process. As Table 2.3 indicates, **KBS** analysis can point out shortcomings at all stages. Thus, as development progresses, there are usually changes in requirements, concepts, specifications, organizing structures and rules.

2.2. KBS Analysis vs Traditional Algorithm Analysis

The characteristics of traditional algorithm analysis are much different from knowledge base system analysis. A brief discussion of the differences between **KBS** analysis and traditional algorithm analysis may be useful.

2.2.1. Characteristics of Traditional Algorithm Analysis

We discuss the characteristics of traditional algorithm analysis in terms of computational model, human factor, domain, implementation, analysis criteria and analysis methods. It is clear that computational theory deals realistically with the quantitative aspects of computing and develops a general methodology, which measures the complexity of computing functions. In order to analyze the complexity of algorithms, the theory must define some notations and concepts of complexity, then discuss the difficulty of a given problem by the existence or absence of an efficient algorithm for that problem.

Computational model : In the field of traditional algorithm analysis, we use the Turing machine (TM) model to determine upper bounds on the space and time necessary to solve a given problem. And usually, we also focus on some particular operations, such as comparison and addition, whose execution time in the computational model can be bounded by a constant.

Human factors : Traditional algorithm analysis is human independent in the sense that we are only interested in analyzing the particular algorithm. Based on the Turing machine model and predefined elementary operation, we can analyze and derive some results from the performance of the algorithm, which is totally independent of those who developed this algorithm. In short, the conclusion of traditional algorithm analysis does not depend upon who designed it, who used it and who analyzed it.

Domain : Traditional algorithm analysis is domain independent. For example, the bubble sort algorithm can be used in any application domains without affecting its complexity. The characteristics of particular domain do not influence the result of algorithm analysis.

Implementation : Algorithm analysis is implementation independent, meaning that it does not require particular hardware and software for implementation. For instance, the result of sorting algorithm analysis does not depend on machines or programming languages.

Criteria : The two major factors are used to judge the goodness of an algorithm are time complexity and space complexity. In each case, it is common to consider average and worst case performances.

Method : When we analyze the efficiency of an algorithm, we define and use the asymptotic notation, such as "the order of n^2" notated as O(n^2), operations on asymptotic notations and asymptotic recurrences and so on. Then we employ quantitative mathematical methods to give exact or probabilistic numerical values for estimating algorithm performance.

2.2.2. Characteristics of KBS Analysis

Computational model : The intended models of knowledge base systems are cognitive computational models. Cognitive computational models should capture some characteristics of intelligent cognitive behavior, which are discrete, symbolic and qualitative. Besides, the cognitive computational model is characterized by a collection of general strategies that use encoded knowledge in such a way that the complexity of computation inherent in certain tasks is minimized. A variety of cognitive computational models which are implemented in knowledge base systems. In diagnostic knowledge base systems, for instance, there are certainty factor model from medicine [77], subjective probabilistic model [21,22], general set covering model [66], and model for diagnosis by using "First principles" [69] and so on. These models are much simplified cognitive models, or just attempted cognitive models. A typical cognitive model should be a mental model which is composed of basic data sources, motivational concepts, conscious experience, unconscious material, external stimuli and a socio-cultural context.

Human factor : **KBS** analysis is human dependent. The evaluators could be domain experts, knowledge engineers, end users and researchers. The **KBS** analyses vary, depending on who is making them. Funding agencies may wish to see a **KBS** system performing in the best possible light. Knowledge engineers probe for deficiencies and ways want to convinced that the knowledge base system represents an improvement over procedures they currently use. And researchers in AI and knowledge base systems are always analyzing and comparing alternative systems and techniques, which help them test new ideas, both their own and those of their peers. Besides, the criteria and methods taken for effectiveness analysis of knowledge base systems depend on who is doing them.

Domain : Knowledge base systems are usually designed for limited or specific task domain. It is natural to analyze a **KBS** according to the domain scope designed. **KBS** analysis is closely related to the particular domain and application.

Implementation : There are many different ways to implement a knowledge base system; **KBS** analysis is related to and restricted by the particular implementation. For instance, the knowledge can be represented by first order logic, production rules, scripts or frame. The inference methods could be resolution principle, conflicting resolution, generate-test, theorem proving or procedural method. And a lot of knowledge base shells are available today. The different kinds of knowledge representations, different inference engine and different shell will influence the performance of the implemented knowledge base system.

Criteria : For many of the application domains encoded by knowledge base systems, it is impossible to identify an answer that is "absolutely correct" for any given problem. During the Mycin development, a well-known experiment (the bacterium experiment) was designed to analyze the effectiveness of the Mycin system. To deal with the lack of an absolute criterion of correctness, it was decided that a team of experts would be polled to establish the "correct" answer for each question. The criteria are the correct responses given by a human expert for the same questions. The criteria to judge the effectiveness of knowledge base systems are very hard to define as they are dependent on the particular problem and particular task domain. The complexity results (time and space usage) in traditional algorithm analysis are given relatively minor consideration.

Methods : Because the field of knowledge base systems is so young, there is no standard set of methods for **KBS** analysis. Generally speaking, we should use the combination of symbolic, qualitative and quantitative analysis for **KBS** analysis.

Based on our previous discussion, Table 2.4 summarizes the conclusions:

Factor	**Algorithm Analysis**	**KBS Analysis**
Model	computational model	attempted cognitive model
Human being	non-relevant	heavily relevant
Domain	non-relevant	heavily relevant
Implementation	independent	dependent
Criteria	well defined	ill-defined
Methods	numerical methods	combination

Table 2.4 Algorithm analysis vs **KBS** analysis.

Table 2.4 shows the factors which are related to **KBS** analysis. Note that each of these factors is an active research area.

Clancey [14] and many other researchers [2,73] view knowledge base systems as cognitive models and argue that the major difference between **KBS** and conventional program is that **KBS**'s reasoning is qualitative, parallel, distributed and imprecise processing. Validating the computational cognitive model has been a controversial issue for decades [12,20]. Gaschnig and others [7,29] briefly analyze the human being factor, which greatly influence the result of **KBS** analysis. They recognize that **KBS** builders, experts, end users and researchers usually have different objectives as well as subjective views, which make the **KBS** analysis difficult. Considering the domain factor, some researchers, such as Slagle [79], Klein [40], Casey and Laufmann [10,41], focus on methodology about evaluating and selecting appropriate **KBS** application domains, and Prerau [62] discusses the problem features that influence the design of a **KBS**. They argue that the selection of potential applications must be guided by strict consideration not only of the abilities of **KBS** technology and current limitations, but also of real-world testing and analysis. In another report, Freedman [25] and Citrenbaum [13] discuss the implementation problem. They consider that once a problem domain is selected and a set of **KBS** shells is available, we should further distinguish the **KBS** shell candidates and find the best shell for implementing the **KBS**.

In this dissertation, we consider two of the six factors in Table 2.4, namely, criteria and methods. We propose **KBS** and **KB** models, define the criteria (effectiveness) which is well defined and use a formal method to analyze the effectiveness of a **KBS** and **KB**.

3.

CHAPTER 3

RELATED WORKS

In this chapter, we survey various related works about effectiveness analysis of the knowledge base. Since most of the previous work has ignored or slightly touched problems of minimality and termination, this survey is geared toward answering how completeness and consistency problems are defined and how algorithms solve these problems. This survey also discusses the relationships among these previous works and points out their inadequacies.

3.1. Davis's Work (1979)

TEIRESIAS [18] is a system that assists in entering and updating the large knowledge bases used in expert systems [3]. One goal of the TEIRESIAS program is to automate knowledge base debugging in the context of the MYCIN infectious disease consultation system [75]. The TEIRESIAS program was the first attempt to automate the knowledge base checking and debugging process.

TEIRESIAS aids a human expert in monitoring the performance of a knowledge base system. When the human expert spots an error in the program's performance, in either the program's conclusions or its line of reasoning, TEIRESIAS assists in finding the source of the error in the knowledge base by explaining the program's conclusions and retracing the reasoning steps until the faulty (or missing) rule is identified. At this point, TEIRESIAS assists in knowledge acquisition, modifying faulty rules or adding new rules to the knowledge base.

During the execution of a knowledge base system (such as MYCIN), the domain expert can call TEIRESIAS to indicate and trace a deficit, or bug in the knowledge base. The procedure includes the following steps :

1. Stopping the performance of the knowledge base system when the human expert identifies an error;
2. Working backward through the inferences in the knowledge base system that led to the error, until the bug is found;
3. Helping the expert fix the bug by adding to or modifying the knowledge base.

This process relies heavily on rule models. TEIRESIAS's rule models are empirical generalizations of subsets of rules, indicating commonalities among the rules in that subset. The knowledge about the principal contents of the domain rules represented in the rule models is used by TEIRESIAS to build expectations or suggestions for identifying information missing from the new rule.

The first program which attempts to debug a knowledge base system is TEIRESIAS, but TEIRESIAS does not systematically check a knowledge base as it was initially established. Rather, it assumes the knowledge base was nearly "complete" and the knowledge base debugging occurred in the setting of a problem-solving session or testing session.

3.2. Suwa's Work (1982)

The problem of logical checks for completeness and consistency in a rule-based expert system was recognized first by Suwa, Scott, and Shortliffe in 1982 [85, 86]. They informally define the problem of completeness and consistency and describe a Rule Checking

Program for use with ONCOCIN, an EMYCIN-like rule-based expert system for oncology [76].

Definitions for Completeness and Consistency Problems

Suwa, Scott and Shortliffe do not give formal definitions for completeness and consistency of a knowledge base system. Instead, they describe what will happen if the knowledge base system is not complete or consistent.

For the completeness problem, they point out:

> Incompleteness of the knowledge base is the result of missing rules. The missing rules mean that a situation exists in which a particular result is required, but no existing rule can succeed and produce the desired result.

For the consistency problem, Suwa, Scott and Shortliffe say :

> Inconsistencies in the knowledge base appear as conflicting, redundant and subsummed rules :
>
> Conflict : Two rules succeed in the same situation but with conflicting results.
>
> Redundancy : Two rules succeed in the same situation and have the same result.
>
> Subsumption : Two rules have the same results but one contains additional restrictions on the situations in which it will succeed. Whenever the more restrictive rule succeeds, the less restrictive rule also succeeds, resulting in redundancy.

In the ONCOCIN expert system, a rule's context and condition together describe the situations in which it applies. Suwa assumes that there should be a rule for each possible combination of condition parameters. Based on this pragmatic assumption, the Rule Checking Program can hypothesize missing rules. The context and condition of two rules can be examined to determine if there are situations in which both can succeed. If so, and the rules conclude different values for the same parameter, they are in conflict. If they conclude the same value for the same parameter, they are redundant. If they are the same except that one contains extra condition clauses, then one subsumes the other.

Rule Checking Program

Suwa notices that these definitions of incompleteness and inconsistency simplify the task of checking the knowledge base. The Rule Checking Program is designed for finding missing rules, conflicting rules, redundant rules and subsummed rules. The rules of a knowledge base can be partitioned into disjoint subsets, each of which concludes about the same parameter in the same context. The resulting rule sets can be checked independently. To check a set of rules, the Rule Checking Program works as follows:

1. Find all attributes used in the conditions of these rules;
2. Make a table, enumerating and displaying all possible combinations of condition attribute values and the corresponding values which will be concluded for the action attribute;
3. Check the tables for conflict, redundancy, subsumption, and missing rules; then display the table with a summary of any potential errors that were found.

In summary, Suwa's work assumes that all possible combinations of attributes are meaningful. Therefore, if the number of attributes is large, the system will suggest a very large number of missing rules. This will lead to computational intractability.

3.3. Nguyen's Work (1985)

The work of Nguyen, Perkins, Laffey and Pecora [50-52] is an extension of the Rule Checking Program designed by Suwa (see previous subsection). They describe a program called CHECK which works with the Lockheed Expert System (LES). The main differences between Nguyen's work and Suwa's work are:

1. CHECK is applied to the entire set of rules, not just the subsets which determine the value of each attribute;
2. Nguyen's work covers some additional problems in knowledge base checking,

such as unreachable conclusions, dead-end IF condition, dead-end goals, unnecessary IF condition, unreferenced attribute value;

3. The program CHECK produces dependency charts and detects any circular rule chains.

Definitions for Completeness and Consistency Problems

Nguyen claims that they have found four situations indicative of incompleteness, and any one of these situations might indicate that there is a rule missing. The four situations are :

1. Unreferenced attribute values. Some values in the set of possible values of an object's attribute are not covered by any rule's IF conditions.
2. Illegal attribute values. A rule refers to an attribute value that is not in the set of legal values.
3. Unreachable conclusions. A conclusion of a rule neither matches a goal nor matches an IF condition of another rule.
4. Dead-end IF conditions and dead-end goals. A data of a rule is neither askable from the user, nor matched by a conclusion of one of the rules in the rule set.

For the consistency problem, Nguyen identifies two more situations than Suwa does. In addition to the conflicting, redundant, and subsummed rules, these situations are:

1. Unnecessary IF conditions. Two rules contain unnecessary IF conditions if the rules have the same conclusions, an IF condition in one rule is in conflict with an IF condition in the other rule, and all other IF condition in the two rules are equivalent.
2. Circular rules. A set of rules is circular if the chaining of these rules forms a cycle.

CHECK Program

The CHECK program assumes that rules are naturally separated by "subject categories," a group of related rules kept together for documentation. Rules are checked against all others in the same subject category and all others that have the same goal. To check a set of rules, the CHECK program performs five steps :

step 1. Each IF and THEN clause of every rule in the set is compared to the IF and THEN clauses of every other rule in the set. The comparison of one clause to another results in a label of SAME, DIFFERENT, CONFLICT, SUBSET, or SUPERSET being stored in a two dimensional table;

step 2. Using the information in step 1, the program is to deduce the relationship between the rules in the set;

step 3. Output the information about redundant, conflicting, subsummed, unnecessary attribute rules;

step 4. Find unreachable, dead-end goal rules;

step 5. Generate a dependency chart for further analyzing circular rules.

Nguyen's CHECK program can identify some incomplete and inconsistent problems in the knowledge base by looking for redundant rules, conflicting rules, subsumed rules, unnecessary IF conditions, and circular rule chains. However, Nguyen's work does not further analyze the circular rules. Some circular rules form a cycle, while others may not form an actual cycle (see Chapter 7). Also, this work lacks theoretical consideration. What we need is a formal treatment of effectiveness analysis.

3.4. Stachowitz's Work (1987)

Stachowitz, Combs, and Chang define the evaluation of knowledge base systems as the verification, validation and testing of a knowledge base system [82-84]. They have partially implemented the Expert System Validation Associate (EVA) at the Lockheed AI Center at Austin, Texas. The EVA is designed to be a metaknowledge-based system shell. It consists of meta-knowledge base and eight checking components. The meta-knowledge base contains some information about the inheritance relation and constraints. The eight checking components are as follows:

1. Logic Checker : To check whether the rule base of an application is consistent

and "numerically complete;"

2. Extended Logic Checker : To check for inconsistencies and conflicts caused by generalization hierarchy, incompatibility or synonymy.
3. Structure Checker : To check rule reachability, redundancy and cycles.
4. Semantics Checker : To check the semantic constraints for consistency.
5. Omission Checker : To check the missing rules.
6. Rule Proposer : To propose some rules which are possibly missing.
7. Behavior Verifier : To check the external input/output interfaces and internal states of the subsystems of knowledge base system. The verifier is to prove that the intended behavior of the system can be derived from the behaviors of the subsystems and the connection descriptions.
8. Control Checker : To verify whether the control constructions violate the specifications.

The EVA is still under development. Basically, this work is an expansion of Nguyen's work (see previous subsection). The above framework has improved Nguyen's work, but suffers the same problem: it is lacking in the theoretical treatment of the effectiveness analysis problem and thus cannot contribute to the enhancement of knowledge base effectiveness analysis technology.

3.5. Cragun's Work (1987)

Cragun and Stendel [17] use a decision-table-supported processor for checking completeness and consistency in rule-based expert systems. This approach creates a large decision table from the rules of the knowledge base, splits the decision table into subtables with similar logic, checks each subtables for completeness and consistency, and reports any missing rules.

The consistency checking is to find ambiguities and redundancies in the rules of the knowledge base. Ambiguity occurs when the same set of logical conditions satisfies two or more different rules that have different actions. Redundancy occurs where the same logical conditions satisfy more than one rule, but the actions are the same.

The completeness checking is to find logical cases that have not been considered, in other words, missing rules. This means that completeness is present when all possible combinations of logic are addressed by rules in the knowledge base.

The idea of Cragun's work is basically the same as that of Suwa's work. The strong precondition for Cragun's work is that there is a mapping between a set of rules and a decision table. This precondition is the same as the assumption of enumerating all possible combinations of rule parameters by Suwa (see previous subsection). This approach also led to combinatorial complexity.

3.6. Morell's Work, (1988)

Morell proposed using metaknowledge in verification of knowledge-based systems [49]. He described a model of knowledge-based systems. This model is a symbolic manipulation system in which a problem from a particular problem space is encoded and manipulated according to a knowledge base in order to produce a solution (see Figure 3.1). In this figure, **H** represents the encoding of the problem, **R** represents the derived solution and the turnstyle represents the deductive process applied by the inference engine operating on the knowledge base. The solid arrow indicates the method of solution that would be used if the operation were performed manually. The dotted line represents the interpretation mapping, **I**, and its inverse relation, $\mathbf{I}^{-1}$. These mappings assign meaning to the represented problem, the knowledge base and the solution. Morell claimed that verification presumes this mapping because any assertion of inadequacy must be grounded in the actual problem to be solved. A computer solves a problem only if it accurately encodes the problem and produces an acceptable solution.

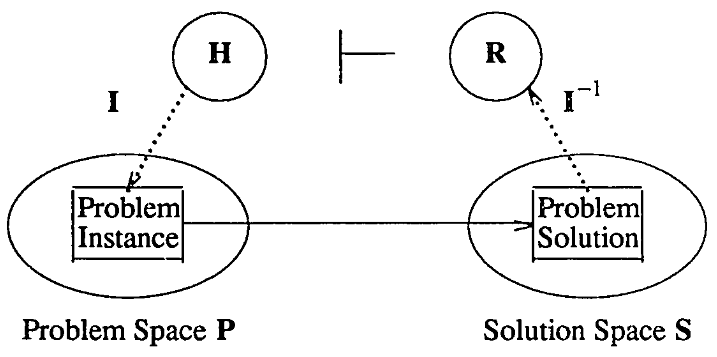

Figure 3.1 A model for knowledge-based systems.

Based on such abstract model, a knowledge-based system can fail in two ways: it can be inconsistent, incomplete, or both. Here, the inconsistency is defined as a characteristic of the interpretation **I**. A knowledge-based system is inconsistent if and only if applying the interpretation function **I** to a state of the knowledge base produces a state that is inconsistent with the modeled world. For instance, in applying **I** at any point, the system is asserting something patently false about the domain it models, and therefore the system is inconsistent.

The system is incomplete if it lacks deductive capability. The formal definition about completeness given by Morell is:

> A deductive system **D** is *complete* for a problem space **P** and an interpretation function **I** if and only if for every $p \in$ **P**, there is a solution $s \in$ **S**.

Incompleteness can arise then from several sources: inadequate expressiveness of the model; inadequate knowledge base; inadequate deductive power.

To check the inconsistent and incomplete knowledge-based system, Morell proposed the use of *metaknowledge* which is defined as knowledge about the knowledge base. According to Morell, an example of metaknowledge would be a listing of reliable sources of medical information for medical knowledge base consistency and completeness checking. Put another way, if the "listing of reliable sources of medical information" says that some disease is "yes," but the corresponding knowledge base system says "no", this indicates the inconsistency of the knowledge base system; and if the "listing of reliable sources of medical information" says that some disease is "yes" or "no," but the corresponding knowledge base system says nothing, this indicates the incompleteness of the knowledge base system.

Remark : Morell's model is a highly abstract model of knowledge-based systems, it is difficult to define clearly. However, it attempts to deal with the effectiveness analysis problem of knowledge-based systems in a theoretical setting.

3.7. Ginsberg's Work (1987, 1988)

Ginsberg [31,32] proposed an approach, called knowledge base reduction, to checking knowledge bases for inconsistency and redundancy. He assumed that the knowledge base is acyclic, therefore it is possible to partition the rules of a knowledge base into different levels. The inference model of KB-Reducer has the three features : (1) monotonicity: if a certain set of input data α causes the inference engine to conclude C, then any superset of α leads to C being asserted; (2) non-selective: there is no conflict-resolution strategy; (3) firing once: for a given data and continually growing contents of working memory, the rules can only be fired one time. The assumptions above seem to be very strong and make the problem less general.

Ginsberg defined the consistency and nonredundancy of knowledge base as follows:

Consistency: A knowledge base is consistent iff there is no way of reaching contradictory assertions from "valid input data."

Nonredundancy: A set of rules R is said to be nonredundant if no $r \in R$ follows from the other member of R, and if no $r \in R$ is such that r can never be satisfied by any valid input set.

For checking the contradictions and redundancy of rules, Ginsberg uses the following procedures:

(1) Ordering the rules of knowledge base;

(2) Using a given list of conflicting information to check contradictions;

(3) Using the superset of some environment of rules to check redundancy;

(4) Backward checking.

The complexity of worst case for such checking is exponential.

3.8. Summary

To summarize, the effectiveness checking of a knowledge base has proven is very important in AI applications, but in general, this involves a repetitive and very complicated process [5,83]. Few works exist today that consider effectiveness checking for a knowledge base system. The theoretical work in this field has just begun.

After surveying the previous related works of by other researchers, we found some serious problems in their results.

1. None of them uses a formal model for describing, analyzing and solving the effectiveness problem of the knowledge base system; their work is either domain related (as MYCIN or ONCOCIN) or expert system shell related (as LES). This problem makes their work lack generality.
2. The problem definitions for completeness and consistency of knowledge base are not precise and clear. Actually, this problem is the logical consequence of lacking a theoretical model.
3. They do not emphasize the minimality and termination problems of knowledge base systems. The issues of minimality and termination are significant problems for developing and using knowledge base in the real world.

The purposes of this dissertation are as the follows:

1. We define that an effective knowledge base system should satisfy the following properties: minimality, termination, completeness and consistency.
2. We will explore and define a general framework for describing and analyzing the effectiveness of a knowledge base system.

3. Based on such models, we will define the minimality, termination, completeness and consistency problems for knowledge base.
4. We will design and analyze a set of algorithms for solving the effectiveness analysis problem of knowledge base.

We attempt to use a formal approach which borrows and uses some concepts and techniques from formal language theory, relational data base theory and Petri net theory, for exploring and developing the theoretical basis for knowledge base effectiveness analysis. Also we want to get some computational results about checking problems. In other words, we want to give a formal ingredient to the knowledge base effectiveness checking problem.

4.

CHAPTER 4

MODELS AND TOOLS

In this chapter, we develop our theoretical models for knowledge base system and knowledge base respectively, as well as some tools which we will need. We first discuss the power and structure of knowledge. We subscribe to Clancey's point of view [14] that knowledge bases can be viewed as qualitative models. The basic idea is to propose a general framework for knowledge base systems and knowledge bases under certain problem settings. Based on these models, we can further explore the effectiveness problem for knowledge base systems in a systematical and computational way.

4.1. The Power and Structure of Knowledge

4.1.1. Knowledge × Search = Complexity

Knowledge is power, knowledge can compensate for lack of search and the search can compensate for lack of knowledge [65]. We propose a formula to illustrate this relationship:

$$\text{Knowledge} \times \text{Search} = \text{Complexity}.$$

A typical example is to open a safe whose lock has ten dials, each with one hundred possible settings, numbered from 0 to 99. If we use a blind trial-and-error search for the correct setting, there are 100^{10} possible settings. We may expect to examine about one-half of these, on the average, before finding the correct one-that is 50^{10}, or fifty billion settings. Suppose, however, that we have some heuristic information: the safe is defective, so that a click can be heard when any one dial is turned to the correct setting. Now each dial can be adjusted independently and does not need to be touched again while the others are being set. The total number of settings that has to be tried is, on average, 10×50, or 500. The task of opening the safe has been altered, by the cues the clicks provide, from a practically impossible one to a trivial one (this example from H. A. Simon) [78]. In the knowledge lies the power which can reduce search efforts tremendously in a problem solving process. The purpose or fundamental principle of knowledge base system building is to represent the knowledge explicitly, reason with it and solve the given problem.

4.1.2. Knowledge = KnowW + KnowHow

The step from data to knowledge is a big one. We can look at data as being static, implicit, interpretable and passive information, and at knowledge as dynamic, explicit, explainable and active information. Generally speaking, knowledge should contain data. Actually, the word "knowledge" is not very well defined in either computer science or psychology. According to one dictionary [23], "knowledge is justified true belief." Each of the words in the definition can be argued; i.e., what is belief, what is true and what is justified? However, in the context of "knowledge base systems," this definition fits rather well. In for representing knowledge, it must be possible to interpreted *propositionally*, that is as expressions in a language with a truth theory [44]. This means that knowledge has a propositional property which is fairly well suited to its intuitive, everyday common sense. Our models for knowledge base systems and knowledge bases are based on this propositional property.

Next, we consider the structure of knowledge. We propose a high level formula: *Knowledge = KnowW + KnowHow.* Knowledge has two components, KnowW and KnowHow. The first component, KnowW contains knowledge elements, the generalized concepts or vocabularies, about objects in terms of "what is that," "when it works," "why it works" and "where is it." This kind of knowledge is called *declarative knowledge*, which is used to focus on categories, space and time properties of objects. In other words, the declarative knowledge is that which the human system can encode it quickly and without commitment to how it will be used. Declarative knowledge is what is deposited in human memory when someone is told something, as through instruction or reading a text book. The second component, KnowHow, contains the particular methods, actions and process sequence which make things work. This knowledge is called *procedural knowledge.* Procedural knowledge includes the relationships among the objects, the heuristic information, rules of thumb, structural and functional reasoning process. Procedural knowledge can only be acquired through the use of declarative knowledge, often after trial and error practice, and is further characterized by the fact that it embodies the knowledge in a highly efficient and use-specific way. An ideal knowledge base system should encode all the necessary KnowW and KnowHow of human experts about the certain, in most cases very narrow, well-circumscribed domain in which we are interested.

4.2. Model for Knowledge Base Systems (KBS)

In this section we propose a framework sufficiently general in nature. This model forms the basis for our macro level investigation of effectiveness. A knowledge base system(**KBS**) over a particular application domain can be described as a system **(U, T, R, F, M)**, the definition is:

Definition 4.1 A *knowledge base system* is a five tuple:

KBS = (U, T, R, F,M),

where

- **U** : application domain universe
- **T** : relational table
- **R** : set of rules
- **F** : set of functional specifications
- **M** : mechanism for the knowledge base system.

An application domain universe (**U**). Any knowledge base system is designed for solving problem in a particular application domain. The application domain universe contain finite knowledge elements which can be notated as

$$\mathbf{U} = \{ u_1, u_2, ..., u_n \}.$$

For instance, in a knowledge base system for medical diagnosis, the knowledge element u can be either system or disease. Generally speaking, the knowledge element u_i is some condition or action, and some element can be both condition and action. A knowledge element u_i is not necessary *atomic* or *singleton*. The application domain universe **U** is symbolic representation for *knowW* in a particular domain.

A relational table(T). After having acquired expertise in some domain, the knowledge engineer can build a relational table **T** which describes the *KnowHow* of some experts in the application domain. The relational table is different from the relational model in database theory in the sense that the element in the table do not have to be atomic; there is no consideration about normal form [88]. We believe that for any application domain, the KnowHow of experts, which implies the relationships among the elements in application domain universe **U**, can be represented by a relational table. The relational table **T** contains finite tuples which can be notated as:

$$\mathbf{T} = \{ t_1, t_2, ..., t_m \}.$$

Each tuple $t_i \in \mathbf{T}$ has some condition elements and action elements. The t_i can be represented as

$$t_i = \{ t_{i,1}, t_{i,2}, ..., t_{i,p} \}$$

where $t_{i,j} = u_k$ or $t_{i,j} \in u_k$, $u_k \in$ U, $1 \leq j \leq p$ and $1 \leq k \leq n$.

A set of rules (**R**). The **R** is a set of rules which are encoded KnowHow about a certain application domain. The format of rules is implementation dependent. It could be frame, first order logic, script, or production rule. The relation and distinction between a relational table **T** and the set of rules **R** are that:

(1) The set of rules **R** is based on and created from the relational table **T**;

(2) The set of rules **R** is a more concise and more compacted representation of KnowHow.

(3) If we view the relational table **T** as "source KnowHow," the set of rules **R** can be referred to as "encoded KnowHow."

In this dissertation, we concentrate on production rule format. The reasons for this choice are twofold : first, the "IF-THEN" structure is a fundamental format for knowledge representations, either explicitly or implicitly; second, so far the majority of applied knowledge base systems are implemented by using rule-based technique [56,92]. The set of rules **R** can be notated as

$$\mathbf{R} = \{ r_1, r_2, ..., r_s \}.$$

For any rule r_i, $1 \leq i \leq s$, its format is as follows:

$$\text{If } (c_1 \text{ and } c_2 \text{ and } ... \ c_l) \text{ then } a$$

or

$$c_1 \wedge c_2 \wedge ... \ c_l \rightarrow a$$

here, the c_j(condition or situation) and a (action or advice), $l \geq j \geq 1$, are more general, more compacted and concise in the sense that c_j and a can be a knowledge element, a subset of some knowledge element or a cluster of some knowledge elements from universe U, respectively.

A set of functional specifications (**F**). **F** represents the functional specifications for a knowledge base system. The functional specifications depend on particular application domain and are derived from project objectives, user requirements and design specifications. In order to check the completeness of a knowledge base system, we have to have complete functional specifications for the **KBS**, otherwise the completeness checking can not be carried out. The functional specifications can be expressed as

$$\mathbf{F} = \{ f_1, f_2, \ldots, f_q \}.$$

A mechanism (**M**). The mechanism **M** includes the processor (inference engine) and interface. The processor uses the knowledge in the selected representation to solve a given problem. The processor could be a general theorem-prover, a pattern matcher, a generate-and-test method, backtracking or a conflicting resolution mechanism. The interface is the facility to handle input and output information. Its task is to facilitate the communication between users and knowledge base. The current trend for developing knowledge base system is that we use a general, domain "independent" reasoning mechanism, for example, a production rule controller or a theorem prover. The mechanism, which is used in this dissertation, is an implicit token propagation net **KBNet** (see later). In this dissertation, we contend that it is not necessary to design a very complicated mechanism to achieve the effectiveness of a **KBS**.

4.3. Model for Knowledge Base (KB)

In this section we present a model for checking the effectiveness of a knowledge base system at the micro level. We define a domain-independent model of knowledge bases and use production rule for representing domain information. In this dissertation we focus on one type of knowledge base in which the conjunction of hypotheses is the solution, explanation, decision or action for a given subset of evidences. This model is similar to Reiter [70], the main difference being that we use the model for checking effectiveness.

The formal definition of the model for knowledge base is as follows:

Definition 4.2 A *knowledge base* is a triple **KB** = (**E, H, R**) where :

(1) **E** a finite nonempty set of evidences $\{ e_1, e_2, ..., e_n \}$

E represents all observable facts, symptoms, conditions or askable findings;

(2) **H** a finite nonempty set of hypotheses $\{ h_1, h_2, ..., h_m \}$

H represents all conclusions, actions, diagnostics or explanations;

(3) **R** a finite nonempty set of IF - THEN rules, $\{ r_1, r_2, ..., r_k \}$.

R represents all associative knowledge, or encoded knowhow.

The rules are special mapping functions, there are three types of rules:

Type I : From Evidences to Hypotheses ($\mathbf{E} \rightarrow \mathbf{H}$), notated as $\mathbf{R_I}$.

Type II : From Evidences × Hypotheses to Hypotheses($\mathbf{E} \times \mathbf{H} \rightarrow \mathbf{H}$), notated as $\mathbf{R_{II}}$.

Type III : From Hypotheses to Hypotheses ($\mathbf{H} \rightarrow \mathbf{H}$), notated as $\mathbf{R_{III}}$.

Given a knowledge base **KB** = (**E, H, R**), **R** represents a set of associative knowledge rules, $\mathbf{R} = \mathbf{R_I} \cup \mathbf{R_{II}} \cup \mathbf{R_{III}}$ and in the format:

IF antecedent THEN consequent

or it can be simplified as

IF **P** THEN **Q**, or in the form $\mathbf{P} \rightarrow \mathbf{Q}$.

Both the antecedent(**P**) and consequent(**Q**) are composed of conjunctive propositions (main concepts or vocabularies) in monotonic logic, and the antecedent(**P**) does not contain any negative propositions, as follows:

$$\bigwedge_{i=1}^{k} P_i \rightarrow \bigwedge_{j=1}^{l} Q_j, \quad \text{where the } P_i, Q_j \text{ are literals.}$$

We also use the following notations:

- LHS(r) : the set of all propositions in the antecedent part of rule r.
- LHS(r).hypo : the set of all hypothesis propositions in the antecedent part of rule r.
- LHS(r).evid : the set of all evidence literals in the antecedent part of rule r.
- RHS(r) : the set of all propositions in the consequent part of rule r.
- |S| : the cardinality of the set S.

- LHS($\{r_i, r_j\}$) = LHS(r_i) $\cup$ LHS(r_j)
- RHS($\{r_i, r_j\}$) = RHS(r_i) $\cup$ RHS(r_j)

Example 4.1 Let a knowledge base be **KB** = (**E, H, R**) where

$$\mathbf{E} = \{ e_1, e_2, e_3, e_4, e_5 \}$$

$$\mathbf{H} = \{ h_1, h_2, h_3, h_4, h_5 \}$$

$$\mathbf{R} = \{ r_1, r_2, r_3, r_4 \}$$

$$r_1 : e_1 \wedge e_2 \rightarrow h_2$$

$$r_2 : e_4 \rightarrow h_1$$

$$r_3 : e_3 \wedge h_1 \rightarrow h_3 \wedge h_4$$

$$r_4 : e_5 \rightarrow \neg h_2$$

$$r_5 : h_2 \rightarrow h_5$$

We have :

LHS(r_1) = $\{ e_1, e_2 \}$,

LHS(r_3).hypo = $\{ h_1 \}$,

LHS(r_3).evid = $\{ e_3 \}$

RHS(r_4) = $\{ \neg h_2 \}$,

$|\mathbf{E}| = 5$, $|\mathbf{H}| = 5$, $|\mathbf{R}| = 5$,

$\{ r_1, r_2 \} \subseteq \mathbf{R}_\mathrm{I}$,

$\{ r_3 \} \in \mathbf{R}_\mathrm{II}$

$\{ r_5 \} \in \mathbf{R}_\mathrm{III}$

4.4. Knowledge Base Net (KBNet)

In this section we introduce a graphical representation for inference structure of a knowledge base. We start with a brief introduction to Petri nets [60,61], since our Knowledge Base Net (**KBNet**) is developed from it. Petri net theory provides a mathematical representation of systems. Petri nets were designed and used for modeling and analyzing the class of discrete event systems with concurrency, parallel, asynchronous nature and nondeterminism.

As Petri net theory finds its way into different research and application areas, there are many extensions and derivations of the original Petri net definition proposed to solve the problems of a specific domain. In this dissertation, we are interested in using Petri net theory to develop Knowledge Base Net (**KBNet**) for rule-based system effectiveness analysis. Intuitively, the places in the Petri net can be used to represent propositions, knowledge items, concepts or vocabularies in the knowledge base. In order to accommodate knowledge representation and reasoning, we have to extend the original Marked Petri Net (**MPN**) in the following way.

4.4.1. Marked Petri Net (MPN)

Definition 4.3 *A Marked Petri Net* is a triple, **MPN** = (**P**, **T**, **A**), where

P is a set of places

T is a set of transitions

A is a set of arrowed arcs, $\mathbf{A} \subseteq \mathbf{P} \times \mathbf{T} \cup \mathbf{T} \times \mathbf{P}$

Graph representation: The circle nodes represent the places; bar nodes represent transitions; the solid arrow is used for transition action.

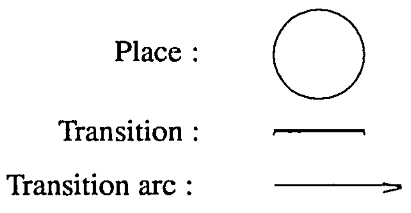

Figure 4.1 Graph representation for **MPN**.

Input Place Set: The input place set of transition t_i is

$$\mathbf{I}_i = \{\, p \mid (p, t_i) \in \mathbf{A} \,\}.$$

Output Place Set: The output place set of transition t_i is

$$\mathbf{O}_i = \{ p \mid (t_i, p) \in \mathbf{A} \}.$$

Markers: The solid dot in a circle node indicates that the predicate represented by the circle node is true.

Mapping Function: The mapping function m is defined as follows, for $p \in \mathbf{P}$,

$$m(p) = \begin{cases} 1 & \text{if a marker in } p \\ 0 & \text{if it is unmarked in } p. \end{cases}$$

Enable Transition: The transition t_i is *enabled* if and only if $m(p) = 1$ for all $p \in \mathbf{I}_i$. In other words, if the t_i is an arbitrary transition, and if there is at least one place p, such that $p \in \mathbf{I}_i$ and $m(p) < 1$, then the t_i is not enabled. In Figure 4.2, since $m(p_2) < 1$, the t_i is not enabled, the place p_3 is unmarked.

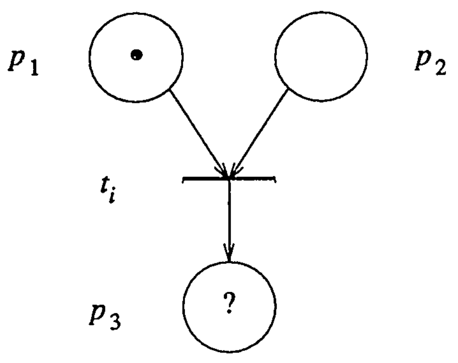

Figure 4.2 The transition t_i is not enabled.

We can find that there is a great similarity between a transition in net theory and a production rule in knowledge base system. Each production rule can be expressed by a transition in a **MPN**, and vice versa. The example in Figure 4.2 can be expressed with the following rule format:

IF (p_1 is true) and (p_2 is true) THEN p_3 is true.

In the Figure 4.1, the dot in the place p_1 means that place p_1 is true.

In Figure 4.3, since $m(p_1) = \mathrm{m}(p_2) = 1$, the transition t_i is enabled, and the predicate p_3 is marked.

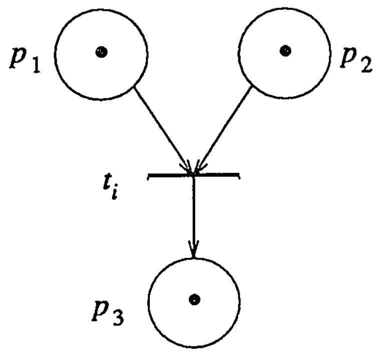

Figure 4.3 The transition t_i is enable.

MPN nets have been developed to enhance the modeling and analyzing power of Petri nets and to offer a formal treatment of individual objects and their relationships in a real world system. A transition t , notated graphically by a bar, describes some kind of relations (causal, implication or relevant) between its input predicates and output predicates. Input places connected to a transition, t, are considered conditions of t; output places connected from a transition, t, are the conclusion of t. The meaning of such kinds of connections can be understood as follows: "if all the conditions of a transition t are satisfied, then the conclusions of t can be derived." In the following discussion, we implicitly use markers and their propagation.

4.4.2. Knowledge Base Net (KBNet)

A rule-based knowledge base can be described by using Marked Petri Net (**MPN**): For a given **KB** = (**E, H, R**), the evidences **E** and hypotheses **H** can be considered as places **P** in **MPN**, an implication of each rule r_i, $r_i \in$ **R**, corresponds to a transition t_i, $t_i \in$ **T**, in **MPN**. For example, the rule r_4 in Example 2.1,

$$r_4 : e_5 \rightarrow \neg h_2$$

the predicates e_5 and $\neg h_2$ are places; the logical implication "$\rightarrow$" of this rule is a transition; and "$e_5 \rightarrow$" and "$\rightarrow \neg h_2$" correspond two solid arcs, which are from e_5 to the transition and from the transition to $\neg h_2$ respectively.

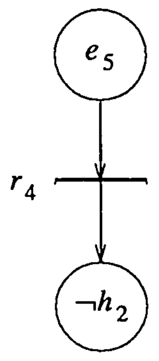

Figure 4.4 The KBNet for rule r_4.

Definition 4.4 *A Knowledge Base Net*, **KBNet** = (**P**, **T**, **A**), is a **MPN**, where

P is a set of evidences **E** and a set of Hypotheses **H**, **P** = **E** ∪ **H**.

T is a set of rules, the transition is the implication of rule, **T** = **R**.

A is a set of directed arcs, **A** ⊆ **P** × **T** ∪ **T** × **P**.

For example, the **KB** in example 4.1 is described by following **KBNet**:

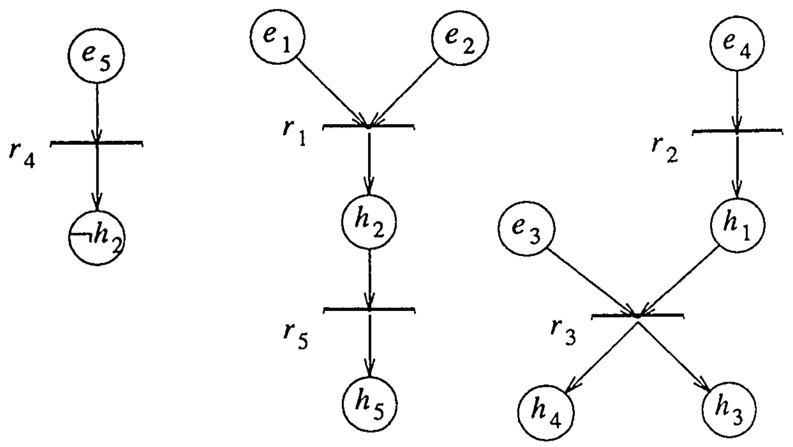

Figure 4.5 The **KBNet** for example 4.1.

For rule r_i, we have

LHS(r_i) = $\mathbf{I}_i$ (Input predicate set for rule r_i)

RHS(r_i) = $\mathbf{O}_i$ (Output predicate set for rule r_i)

We will use the defined **KBNet** for describing and analyzing the characteristics of a given knowledge base system. To describe the behaviors of knowledge base systems, we need some new notations.

Definition 4.5 A rule, $r \in \mathbf{R}$, is able to be *fired* if its left hand conditions (LHS(r) or $\mathbf{I}_r$) can be satisfied.

For example, if e_1 and e_2 are true, then the rule r_1 is firable in example 4.1. Furthermore, after the rule r_1 is fired, the rule r_5 immediately becomes firable in the same example. This constructs a *rule firing sequence* $\{ r_1, r_5 \}$. This leads into the other definition :

Definition 4.6 If rule r_i is fired, after its firing, all rules which are fired construct a *rule firing sequence*, notated as Π, $\Pi = \{ r_i, r_{i1}, r_{i2}, ..., r_{ip} \}$, and $p \geq 1$.

Notice that the p is possibly infinite under certain conditions according to our implicit token propagation mechanism.

4.5. Example

We select a simple example for explaining the concepts that we discussed. Let us use a simple knowledge base system, which is slightly different from Winston's one [91].

4.5.1. BAGGER: Bag Groceries

Suppose we want to develop a knowledge base system to bag groceries in the manner of a grocery-store checkout clerk. We do not expect optimal packing, but we do want our system to know some of the fundamentals of grocery bagging, namely, the large items go in the bottom first, then bag the medium items, and bag the small items last, putting them wherever there is room.

4.5.2. Functional Specifications for BAGGER System

The functional specifications for **BAGGER** system, **F**, can be described as:

F : To bag all available items.

The input space can be partitioned and represented as:

IN_1 : Available items from one type (large, medium or small);

IN_2 : Available items from two types;

IN_3 : Available items from three types.

4.5.3. Rule Set for BAGGER System

We name the packing knowledge base system as **BAGGER**. The necessary knowledge to carry out the bagging job can be captured in the following rules :

r_1 : if there is a large item to be bagged and
    there is a bag with enough room for a large item
  then put the large item in the bag

r_2 : if there is a medium item to be bagged and
    there is a bag with enough room for a medium item
  then put the medium item in the bag

r_3 : if there is a small item to be bagged and
    there is a bag with enough room for a small item
  then put the small item in the bag

r_4 : if there is a large item to be bagged and
    there is no bag with enough room for a large item
  then start a fresh bag

r_5 : if there is a medium item to be bagged and
    there is no bag with enough room for a medium item
  then start a fresh bag

r_6 : if there is a small item to be bagged and
    there is no bag with enough room for a small item
  then start a fresh bag

r_7 : if there is no item for bagging
  then stop.

The rule set for **BAGGER** is **R** = $\{ r_1, r_2, ..., r_7 \}$.

e_1 : there is a large item to be bagged

e_2 : there is a bag with enough room for a large item

e_3 : there is a medium item to be bagged

e_4 : there is a bag with enough room for a medium item

e_5 : there is a small item to be bagged

e_6 : there is a bag with enough room for a small item

e_7 : there is no bag with enough room for a large item

e_8 : there is no bag with enough room for a medium item

e_9 : there is no bag with enough room for a small item

e_{10} : there is no item to be bagged

The evidence set for **BAGGER** is **E** = { $e_1, e_2, ..., e_{10}$ }.

h_1 : put a large item in the bag

h_2 : put a medium item in the bag

h_3 : put a small item in the bag

h_4 : start a fresh bag

h_5 : stop

The hypothesis set for **BAGGER** is **H** = { h_1, h_2, ..., h_5 }. Now, the rules can be represented as follows:

$r_1 : e_1 \wedge e_2 \rightarrow h_1$

$r_2 : e_3 \wedge e_4 \rightarrow h_2$

$r_3 : e_5 \wedge e_6 \rightarrow h_3$

$r_4 : e_1 \wedge e_7 \rightarrow h_4$

$r_5 : e_2 \wedge e_8 \rightarrow h_4$

$r_6 : e_3 \wedge e_9 \rightarrow h_4$

$r_7 : e_{10} \rightarrow h_5$

4.5.4. KBNet for BAGGER

The **KBNet** for **BAGGER** is in Figure 4.6.

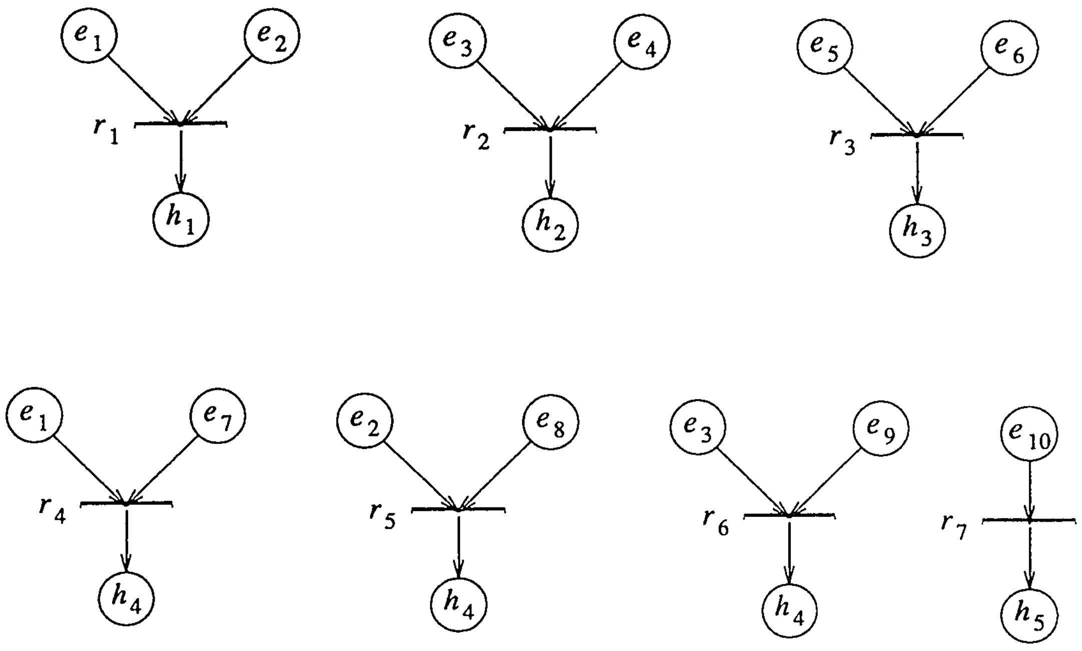

Figure 4.6 The **KBNet** for **BAGGER** system

Summary:

In this chapter, we introduced the model **KBS** for knowledge base systems and the model **KB** for knowledge bases. These models are our framework for analyzing the effectiveness of a knowledge base. We also introduced the tool Knowledge Base Net (**KBNet**), which is used later for analyzing some properties of rule set, minimality and termination problems. Different tools have been used for representation of knowledge base systems. The use of directed and acyclic networks for representing facts in a first-order-logic, has been known in the AI literature [54]. Directed networks have also been utilized to represent belief-networks and probabilistic dependencies [58]. In the logic programming area, some Predicate-Transition Net Model for horn clauses [96] and computational net model [11] are also introduced or developing. However, the **KBNet** we developed here is used for describing a knowledge base system and dealing with minimality and termination problems. The mechanism employed is implicit token propagation and *one time firing*. Given some evidences and/or hypotheses, some rule is fired. The rule is fired only one time unless the evidence or hypothesis is given again. In this chapter we presented a small example, **BAGGER**, which will be used later.

5.

CHAPTER 5

RULE DEPENDENCY AND PROPERTY

The following discussion in this chapter is based on the model for knowledge base : **KB** = (**E, H, R**). Some rules of **R** may have the *dependency* property. Note that if all rules of a knowledge base belong to Type I, **E** $\rightarrow$ **H**, then they are independent. Let us consider the general case in which the knowledge base contains three types of rules.

5.1. Total and Partial Rule Dependency

Given a **KB** = (**E, H, R**), a subset S of rule set **R**, S = $\{ r_1, r_2, ..., r_p \}, p \geq 2$, may have dependent relation. Let us first consider an example.

Example 5.1 Suppose **R** = $\{ r_1, r_2, r_3, r_4 \}$, and the rules are

$$r_1 : \; e_1 \wedge e_2 \rightarrow h_1$$
$$r_2 : \; h_1 \rightarrow h_2$$
$$r_3 : \; e_4 \rightarrow h_3$$
$$r_4 : \; e_3 \wedge h_2 \wedge h_3 \rightarrow h_4.$$

The **KBNet** for this example is as follows:

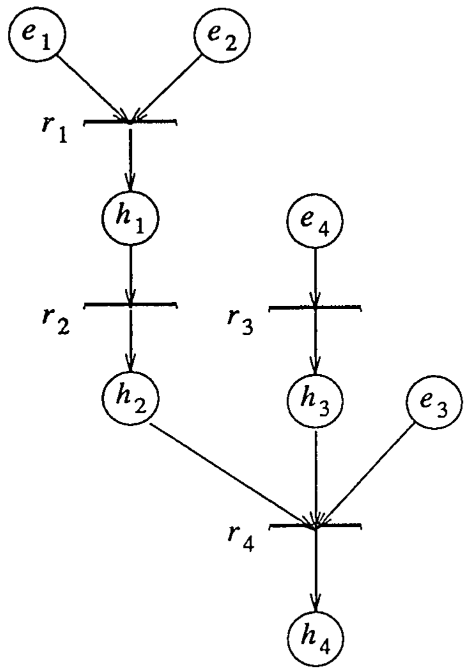

Figure 5.1 The **KBNet** for example 5.1.

In this example, rule r_1 is to be completed before rule r_2 is fired, and rule r_2 is to be completed before rule r_4 can be fired. Furthermore, the example also shows that there are two different situations. First, considering rules r_1 and r_2, after the completion of rule r_1, rule r_2 can be immediately fired. The second situation is that the completion of rule r_2 is one of several preconditions for firing rule r_4. The antecedents of rule r_4 includes e_3, h_2 and h_3, but the completion of rule r_2 just offers h_2. The firing of rule r_4 must wait for the other conditions to be satisfied. The situations above lead up to the following definitions.

5.1.1. Total Dependency

Definition 5.1 Given a **KB** = (**E, H, R**) and a pair of rules, r_i, $r_j \in$ **R**, the rule r_j is said to *totally depend on* a rule r_i, if LHS(r_j) $\subseteq$ RHS(r_i). The total dependent relation of rule r_j on rule r_i is notated by $r_i \propto r_j$, and we say that r_i is a *total predecessor* of r_j, r_j is a *total successor* of rule r_i.

In the example 5.1, the total dependency relation $r_1 \propto r_2$ holds. In this case, r_1 is a *predecessor* of r_2 and r_2 a *successor* of r_1 in the partial ordering. This situation is shown in Figure 5.2,

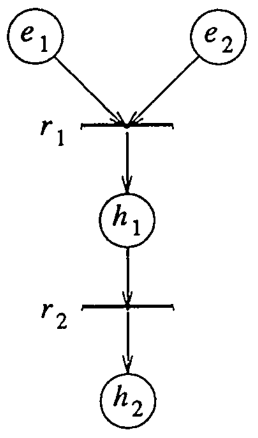

Figure 5.2 The rule r_2 totally depends on rule r_1.

The relation between rule r_2 and rule r_4 in the same example is a different dependent relation. We define it in the following subsection.

5.1.2. Partial Dependency

Definition 5.2 Given a **KB** = (**E, H, R**) and a pair of rules, r_i, r_j $\in$ **R**, i $\neq$ j, rule r_j is said to *partially depend on* a rule r_i, if

(1) RHS(r_i) $\cap$ LHS(r_j) = **I**, **I** $\neq \emptyset$ and

(2) LHS(r_j) - **I** $\neq \emptyset$,

it is notated as $r_i < r_j$, and we say that r_i is a *partial predecessor* of r_j, and r_j is a *partial successor* of rule r_i.

In this definition, the first condition guarantees that rule r_j is partially dependent on rule r_i, and the second condition indicates that the firing of rule r_j still needs other evidences and/or hypotheses to be true. There are three cases under partial dependency:

Case 1: A pair of rules, $r_i, r_j \in \mathbf{R}$, $r_i < r_j$, and $r_j \in \mathbf{R}_{\mathrm{III}}$ (Type III).

For example,

$$r_i : e_1 \land e_2 \rightarrow h_1, \text{ and}$$

$$r_j : h_1 \land h_2 \land h_3 \rightarrow h_4.$$

The **KBNet** for them is:

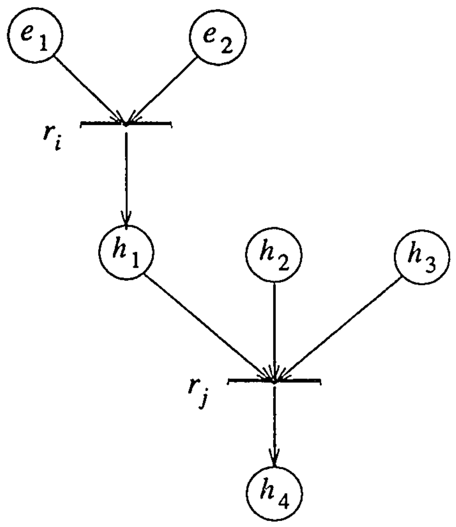

Figure 5.3 KBNet for Case 1.

After the firing r_i, the firing of the rule r_j must wait the h_2 and h_3 to be true.

Case 2: A pair of rules, $r_i, r_j \in \mathbf{R}, r_i < r_j$, and $r_j \in \mathbf{R}_{II}$ (Type II).

For example,

$$r_i : e_1 \wedge e_2 \rightarrow h_1, \text{ and}$$

$$r_j : e_3 \wedge h_1 \rightarrow h_2 \wedge h_3.$$

The **KBNet** for this case is:

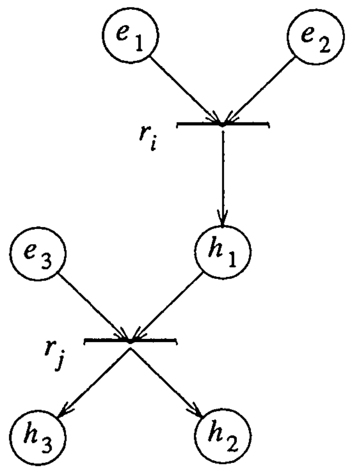

Figure 5.4 KBNet for Case 2.

Here, after firing r_i, the firing of the rule r_j needs one more evidence e_3 to be true.

Case 3: A pair of rules, $r_i, r_j \in \mathbf{R}$, $r_i < r_j$, and $r_j \in \mathbf{R}_{II}$ (Type II).

For example,

$$r_i : e_1 \wedge e_2 \rightarrow h_1, \text{and}$$

$$r_j : e_3 \wedge h_1 \wedge h_2 \rightarrow h_3.$$

The **KBNet** for the case is:

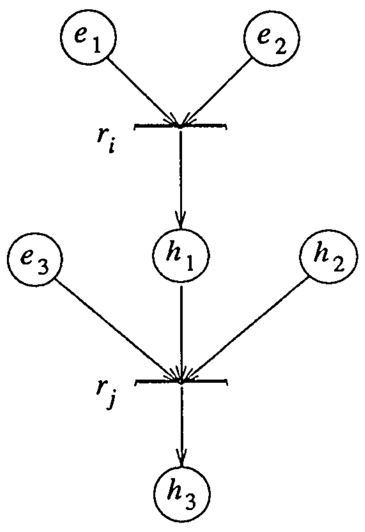

Figure 5.5 **KBNet** for Case 3.

The firing of the rule r_j requests both evidence e_3 and hypothesis h_2 to be true.

In Example 5.1, the partial dependent relation, $r_2 < r_4$ and $r_3 < r_4$ hold. In this situation, both rule r_2 and rule r_3 are predecessors of rule r_4.

5.1.3. Predecessor Set and Successor Set

Considering the predecessors and successors of a rule, we define the notions of the predecessor set and successor set.

Definition 5.3 Let $r \in R$, and $S = \{ r_1, r_2, ..., r_p \} \subset R$, $p \geq 1$. S is a *Predecessor Set* of the rule r, notated by Pre(r), if it satisfies following properties:

(1) For $r_i \in S$, $p \geq i \geq 1$, the relation $r_i \propto r$ or $r_i < r$ holds;

(2) LHS(r).hypo $\subseteq$ RHS(S);

(3) No subset of S satisfies both of (1) and (2).

In example 5.1, Pre(r_2) = { r_1 }, and Pre(r_4) = { r_2, r_3 }. In real problem setting, where the situations could be very complicated, we have the following remarks:

Remark 1: If rule r belongs to Type III, r $\in$ $\mathbf{R}_{III}$, then the firing of the rules in Pre(r) will lead to the firing of the rule r.

Remark 2: If the relation $r_i \propto$ r holds, then $\exists$ Pre(r) = S, $|S| = 1$.

Remark 3: If rule r belongs to Type II, r $\in$ $\mathbf{R}_{II}$, then the firing of the rules in Pre(r) will not necessarily lead to the firing of the rule r, since the firing of rule r needs further evidence(s) to be true.

Remark 4: For a rule r $\in$ **R**, the predecessor set Pre(r) is not necessarily unique.

Similarly, we can define the *Successor set* for a rule r.

Definition 5.10 Let r $\in$ R, and S = { $r_1, r_2, ..., r_p$ } $\subset$ R, p $\geq$ 1. S is a *Successor Set* of the rule r, notated by Suc(r), if it satisfies following property: for $r_i \in$ S, $1 \leq i \leq p$, the relation r $\propto r_i$ or r $< r_i$ holds.

Remark 5: A rule r, r $\in$ **R**, can have total and/or partial successors regardless of its Type.

Remark 6: The Successor set Suc(r) is unique.

The relationship between rule type, predecessor and successor is summarized in the following Table 5.1.

Rule Type	Predecessor	Successor
Type I	No	Total & partial successor
Type II	Partial predecessor	Total & partial successor
Type III	Total & partial predecessor	Total & partial successor

Table 5.1 Relation between rule type, predecessor and successor.

5.2. Rule Dependency Graph

Even though the **KBNet** can give a detailed description of the structures of the rule set, we would like have a higher level way to examine them. In this section, we introduce a graphical representation for inference structure of a knowledge base in terms of rules, which we call the Rule Dependency Graph. The main difference between our Rule Dependency Graph and Ullman's Connection Graph [88] is that we use the total and partial dependent relations to construct our graph.

Definition 5.5 A *Rule Dependency Graph*(**RDG**) is a directed graph,

$$\mathbf{RDG} = (\mathbf{V}, \mathbf{E_T}, \mathbf{E_P}), \text{ where}$$

(1) **V** is a finite set of rules, $\{ r_1, r_2, ..., r_s \}$.

(2) $\mathbf{E_T}$ is a totally dependent relation on **V**, solid arrowed edge from rule r_i to r_j if and only if rule r_j is totally dependent on rule r_i. In other words, edge(r_i, r_j) $\in \mathbf{E_T}$ if and only if $r_i \propto r_j$.

(3) $\mathbf{E_P}$ is a partially dependent relation on **V**, dotted arrowed edge from rule r_i to r_j if and only if rule r_j is partially dependent on rule r_i. In other words, edge(r_i, r_j) $\in \mathbf{E_T}$ if and only if $r_i < r_j$.

Example 5.2 Given a set of rules as follows:

$r_1 : e_1 \wedge e_2 \rightarrow h_1$	$r_2 : e_3 \wedge h_1 \wedge h_3 \rightarrow h_4$
$r_3 : e_4 \wedge h_1 \rightarrow h_2$	$r_4 : e_6 \wedge h_2 \rightarrow h_3$

The corresponding **RDG** is shown in Figure 5.6:

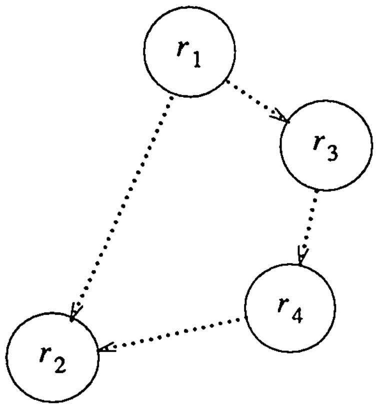

Figure 5.6 Example Rule Dependency Graph (**RDG**)

The arcs are dotted, since $r_1 < r_3$, $r_3 < r_4$, $r_4 < r_2$ and $r_1 < r_2$.

If we take the **BAGGER** example (introduced in Chapter 4), its **RDG** is a set of isolated rule nodes as follows:

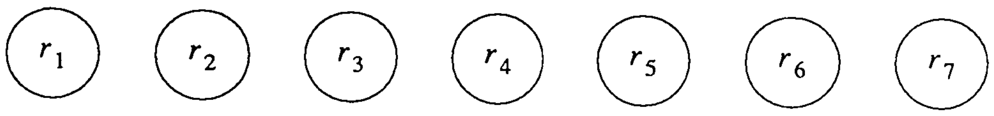

Figure 5.7 The **RDG** for **BAGGER** system

5.3. Restricted Bag and Dependent Closure

In order to recognize the cycles in a knowledge base, we give the following specific notations : *restricted bag* and *dependent closure.*

5.3.1. Restricted Bag

Bag theory [59] is a natural extension of set theory. A bag, like a set, is a collection of elements over some domain. Unlike a set, bags allow multiple occurrences of elements. In set theory, an element either is or is not a member of a set. In bag theory, an element may be in a bag zero times (not in the bag) or one time, two times or any specified number of times.

Definition 5.6 Given a domain, a *bag* is a collection of elements over this domain. Any element of the bag may have n occurrences, n is a positive integer [59].

For checking the terminating condition of a knowledge base, we develop a *restricted bag*, which only allows one of the elements in a bag to have at most two occurrences in the bag.

Definition 5.7 A *restricted bag* **B** is a bag which only allows one of its elements to have two occurrences.

We use the number of occurrences function, notated as #, for defining the number of occurrences of an element in a restricted bag. For an element x and a bag **B**, we denote the number of occurrences of x in **B** by #(x, **B**). As examples, consider the following bags over the domain $\{ r_1, r_2, r_3, r_4 \}$:

$$\mathbf{B}_1 = \{ r_1, r_2, r_3 \}$$

$$\mathbf{B}_2 = \{ r_2, r_3 \}$$

$$\mathbf{B}_3 = \{ r_2, r_3, r_3 \}$$

$$\mathbf{B}_4 = \{ r_1, r_1, r_3, r_3 \}$$

According to our definition 5.6, $\mathbf{B}_1$, $\mathbf{B}_2$ and $\mathbf{B}_3$ are restricted bags, but the $\mathbf{B}_4$ is not a restricted bag because $\mathbf{B}_4$ has two elements whose number of occurrences is two.

5.3.2. Dependent Closure

According to rule dependency, either total dependency or partial dependency, we find that after having some rule's firing, some other rules either have to be fired or may be fired. This phenomenon is worth further study in order to check the terminating condition for a given knowledge base.

Definition 5.8 Let r $\in$ **R**, the *total dependent transitive closure* of the rule r, notated by T_closure(r), is a restricted bag and constructed as follows:

(1) r $\in$ T_closure(r);

(2) while T_closure(r) is not a restricted bag and $\exists$ rule $r_j \in$ **R** can be checked do: if $r_i \in$ T_closure(r), $r_j \in$ **R** and $r_i \propto r_j$, then $r_j \in$ T_closure(r).

Note that since the T_closure(r) is a restricted bag and rule set **R** is finite, the process of constructing T_closure(r) will stop until either no more rule in **R** can be found or the restricted bag condition is satisfied. Intuitively, the T_closure(r) contains all the rules that have to be fired after rule r is fired. We give similar definitions for semi-total dependent transitive closure and partial dependent transitive closure.

Definition 5.9 Let r $\in$ **R**, the *semi-total dependent transitive closure* of the rule r, notated by ST_closure(r), is a restricted bag and constructed as follows:

(1) r $\in$ ST_closure(r);

(2) while ST_closure(r) is not a restricted bag and $\exists$ rule $r_k \in \mathbf{R}_{\mathrm{III}}$ can be checked do: if $r_i \in$ ST_closure(r) or If S $\subset$ ST_closure(r), $r_k \in \mathbf{R}_{\mathrm{III}}$ and $r_i \propto r_k$ or S = Pre(r_k), then $r_k \in$ ST_closure(r).

Intuitively, the ST_closure(r) contains all the rules that could be fired after the rule r is fired. The main difference between T_closure(r) and ST_closure(r) is that for $r_i \in$ T_closure(r), $|\mathrm{Pre}(r_i)| = 1$, but for $r_i \in$ ST_closure(r), $|\mathrm{Pre}(r_i)| \geq 1$.

Definition 5.10 Let r $\in$ **R**, the *partial dependent transitive closure* of the rule r, notated by P_closure(r), is a restricted bag and constructed as follows:

(1) r $\in$ P_closure(r);

(2) while P_closure(r) is not a restricted bag and $\exists$ rule $r_k \in \mathbf{R}_{II}$ can be checked do: if $r_i \in$ P_closure(r) or If S $\subset$ P_closure(r), $r_k \in \mathbf{R}_{II}$ and $r_i < r_k$ or S = Pre(r_k), then $r_k \in$ P_closure(r).

Intuitively, the P_closure(r) contains all the rules that may be fired after the rule r is fired.

Example 5.3 Given a set of rules :

$r_1 : e_1 \wedge h_1 \rightarrow h_2 \wedge h_3$

$r_2 : h_2 \rightarrow h_4$

$r_3 : h_3 \rightarrow h_5$

$r_4 : h_4 \wedge h_5 \rightarrow h_1$

Its **RDG** is :

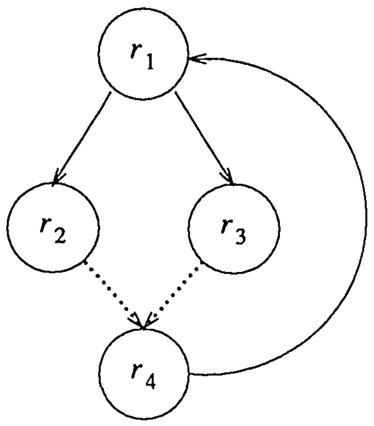

Figure 5.8 Rule Dependency Graph for example 5.3.

The different dependency transitive closures of rule r_1 are :

T_closure(r_1) = { r_1, r_2, r_3 }.

ST_closure(r_1) = { r_1, r_2, r_3, r_4 }.

P_closure(r_1) = { r_1, r_2, r_3, r_4, r_1 }.

5.4. Rule Closure

Definition 5.11 Suppose that $r_1, r_2, ..., r_p$ are rules in R, $r_1 \propto r_2 \propto, ..., \propto r_p$, p ≥ 2, and there does not exist $r_k \in$ R such that $r_p \propto r_k$. Then we have *rule closure* of r_1 as follows:

$$\text{LHS}(r_1) \overset{*}{\Rightarrow} \text{Y}, \text{Y} = Y_1 \cup Y_2 \cup ... \cup Y_p, \text{here } Y_i = \text{RHS}(r_i), 1 \le \text{i} \le \text{p}.$$

We notate the *rule closure* of rule r as r_closure(r).

Example 5.4 Given the following rules:

$r_1 : e_1 \wedge e_2 \rightarrow h_1 \wedge h_2$

$r_2 : h_1 \rightarrow h_3$

$r_1 : h_3 \rightarrow h_5$

We can have:

$\text{LHS}(r_1) \overset{*}{\Rightarrow} Y_1 \cup Y_2 \cup Y_3$

$Y_1 = \{ h_1, h_2 \}$

$Y_2 = \{ h_3 \}$

$Y_3 = \{ h_5 \}$

the r_closure(r_1) = { h_1, h_2, h_3, h_5 }

Intuitively, the rule closure of a rule r contains all hypotheses which can be asserted by firing rule r.

Summary

In this chapter, we studied rule dependency and its property for a rule set **R**, based on the **KB** model. We introduced some new concepts: *total dependency, partial dependency, predecessor set* and *successor set.* We also introduced Rule Dependency Graph (**RDG**), which is used to describe the inference structure of a knowledge base. We further defined *restricted bag, dependent closure (T_closure, ST_closure* and *P_closure)* and *rule closure.* These concepts and notations developed in this chapter will be used for analyzing the effectiveness of a knowledge base.

6.

CHAPTER 6

MINIMALITY PROBLEM

We can now begin to analyze the effectiveness of knowledge base systems. The effectiveness of a knowledge base system, as we defined, includes minimality, termination, completeness and consistency. We study these issues in Chapters 6, 7, 8 and 9. Based on the models **KBS** and **KB**, in each chapter we will first clarify and define the problem, and then design some algorithms to solve the problem.

In this chapter, we discuss the minimality problem. In terms of the domain, under certain circumstances, we can say: *one picture is worth 1,000 words*, but in another circumstance we could say: *one word is worth 1,000 pictures*. The knowledge contained in *one picture* or in *one word* is represented in the sense of minimal size; the *minimum of knowledge* for solving problems on some domain is heavily context-dependent. The minimality problem of a knowledge base system, as we discuss here, considers the size of knowledge base systems and knowledge bases. We discuss this problem from **KBS** and **KB** levels.

6.1. Consideration at KBS level

Based on the **KBS** model for a knowledge base system, **KBS** = **(U, T, R, F, M)**, in which the size of a **KBS** at least includes the following items:

(1) total number of knowledge elements in the application domain **U**;

(2) total number of rules in rule set **R**;

(3) the size of mechanism **M**.

The size of a knowledge base system is also directly related to the functional specifications. Even when we consider that the mechanism **M** is an independent component of the **KBS** and ignore it, the minimality problem is still very hard to define. There are many reasons to explain the difficulty.

First, from philosophical and economical points of view, a knowledge base system is *minimal* if the size of the **KBS** is the smallest one and the **KBS** contains all necessary information for solving the required problem. Even considering a narrowed domain, it still depends on many factors: the user's expectation for the system designed, the format of knowledge representation used, the techniques available and the features of the problem to be solved, and so on. It is difficult to define the *minimal* from these perspectives, either numerically or symbolically.

Second, the size of a knowledge base system, $|\mathbf{U}|+|\mathbf{R}|+|\mathbf{M}|$, depends on the desirable functional specifications **F**. If we want to design a knowledge base which encodes truly encyclopedic knowledge, such a system will contain millions of knowledge elements and rules, see that Lenat et al., have worked in developing such encyclopedic (or common sense) knowledge bases [42, 43]. According to Lenat's recent report [43], the size of such a massive knowledge base is on the order of 10^8 axioms. Note that this number did not include the size of the mechanism. If we design a knowledge base system for solving a simple problem in a very narrowed domain, the size of the system, the number of knowledge elements and rules, could be quite small.

Third, the knowledge elements (class names or concepts) of an application domain **U** may be defined arbitrarily. A single knowledge element (class name or concept) may stand for something very complex, such as "state of the patient," "observation of system," or for something quite simple, such as "sex of the patient," "position of the component." As we mentioned before, a knowledge element in an application domain **U** is not necessarily an "atomic" object. Besides, the value of the instances of the knowledge element may take on continuous values (any real number), binary values (true or false), certainty factor (0, 0.1, ..., 0.9, 1), fuzzy value (from 0 to 1) or probabilistic value p ($1 \geq p \geq 0$). The number of the instances depends on the particular knowledge element designed.

What we can do is to give some empirical analysis from successful or publicly accepted knowledge base systems. Let us use MYCIN, XCON and INTERNIST as examples to look at the size of the knowledge base systems:

	# of Class names	# of rules (links)
MYCIN	20	1000
XCON	100	6000
INTERNIST	680	2600 (with 50,000 links)

Table 6.1 Empirical data for **KBS** size

If we check the characterization of the size of knowledge base systems built routinely in the late 1980's [4], we can get some empirical upper bounds as follows:

# of Class names	# of rules (links)
less than 1000	100s to 1000s

Table 6.2 Empirical upper bounds for **KBS**

Note that such empirical analysis did not consider the size of mechanism, and did not explicitly account for the relationship between functional requirements and system size. The method of the empirical analysis seems closer to our **KB** level analysis for the minimality problem.

6.2. Minimality and Redundant Rules

Consideration of minimality at the **KB** level is in terms of the cardinality of the rule set **R**, and the cardinality of knowledge elements for each rule in **R**. In this section, we identify the varieties of redundant rules which affect the minimality of a knowledge base. One major cause which prevents a **KB** from being minimal is that the **KB** contains some redundant rules. Considering the reliability and efficiency of a **KB**, the redundant rules could be important. But these issues are beyond the scope of this dissertation, and thus we do not discuss them here.

6.2.1. Self redundant rule

Our model is focused on rule-based systems; we do not allow recursive rules. If a rule in a knowledge base looks similar to the one in the following example, we call this rule as *self redundant rule*.

Example 6.1: Given a **KB** = (**E, H, R**), and r ∈ **R**,

$$r : e_1 \wedge e_2 \wedge h_2 \rightarrow h_2 \wedge h_3$$

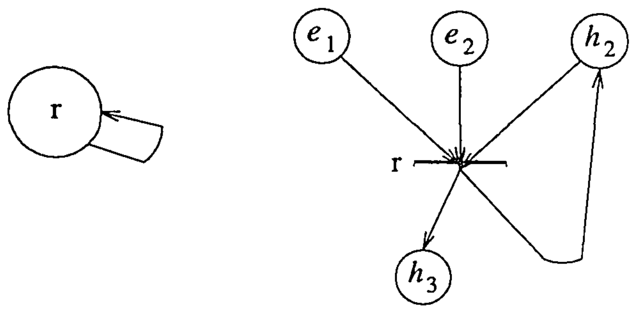

Figure 6.1 KBNet and **RDG** for self redundant rule r.

The rule r contains a redundant element h_2 on its right hand side. The effect of this rule is the same as the following one in a production system:

$$r : e_1 \wedge e_2 \wedge h_2 \rightarrow h_3.$$

Definition 6.1: Given a **KB** = (**E, H, R**), if there is some rule, r ∈ **R**, such that

$$\mathrm{LHS(r)} \cap \mathrm{RHS(r)} \neq \varnothing,$$

the rule r is called as *self redundant rule.*

If the rule r in example 6.1 is semantically correct, it does not cause any logical problem. If we consider the size of each individual rule, it will waste some space for storing one extra knowledge element 'h_2' on its right hand side. But, some problem may arise if we want to modify such rules.

For example, if we intend to modify rule r in Example 6.1 into

$$r_{intend} : e_1 \wedge e_2 \wedge h_1 \rightarrow h_3,$$

there is a possibility that we may forget to eliminate the original redundant element 'h_2' in the right hand side of this rule r, so the resulting rule becomes

$$r_{result} : e_1 \wedge e_2 \wedge h_1 \rightarrow h_2 \wedge h_3.$$

This may cause strange behavior of the knowledge base, because the resulting rule is not our intended one.

6.2.2. Redundant rule

Consider the following example.

Example 6.2 Given a **KB** = (**E, H, R**), and $r_i, r_j \in$ **R**, i ≠ j and

$$r_i : e_1 \wedge h_1 \rightarrow h_3$$

$$r_j : e_1 \wedge h_1 \rightarrow h_3.$$

The corresponding **RDG** and **KBNet** are shown in Figure 6.2.

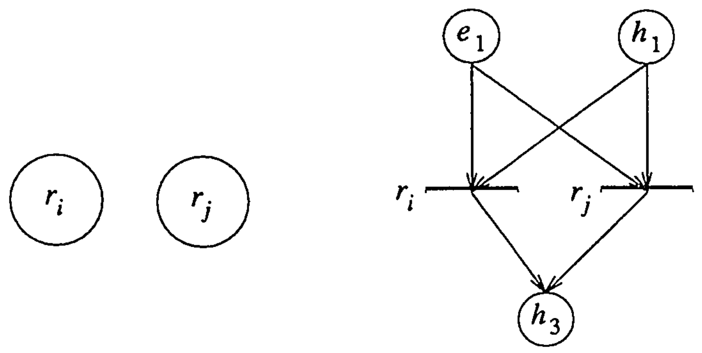

Figure 6.2 RDG and **KBNet** for example 6.2.

Definition 6.2: Given a **KB** = (**E, H, R**), if there is a pair of rules, $r_i, r_j \in$ **R** and i ≠ j, such that

(1) LHS(r_i) = LHS(r_j) and

(2) RHS(r_i) = RHS(r_j)

then one of the rules is called *redundant.*

The two rules, r_i and r_j, succeed in the same conditions and have the same results. Obviously, one of them is redundant. This kind of redundancy in a knowledge base does not necessarily cause logical problems, since in a system, with a particular mechanism, the first successful rule may be the only one to succeed. When we look at the redundant rules from other perspectives, some problems may arise:

Case 1: If we modify the knowledge base and decide to delete such a rule, there is a possibility that we just simply delete rule r_i, and the r_j remains in the knowledge base, where it is a potential hazard.

Case 2: If we revise the knowledge base and modify such rule, there is also a possibility that we just simply modify one of the two rules and the other one is left unchanged. This also may cause some problems.

6.2.3. Subsumed Redundant Rules

There are two kinds of subsumed redundant rules:

Case I: The two rules have the same conclusions, but one contains additional constraints on the situations in which it will succeed.

Example 6.3: Given a **KB**= (**E**, **H**, **R**), and $r_i, r_j \in$ **R**, i $\neq$ j and

$$r_i : e_1 \rightarrow h_4 \wedge h_5$$

$$r_j : e_1 \wedge h_1 \rightarrow h_4 \wedge h_5.$$

From the dependent point of view, there is no precedent relation between the two rules, r_i and r_j. The corresponding **RDG** contains two independent rules (circles).

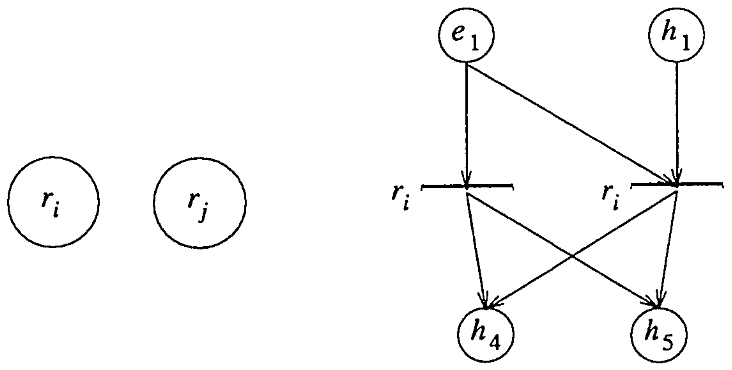

Figure 6.3 **RDG** and **KBNet** for example 6.3.

In this example, we say that rule r_i subsumes rule r_j because rule r_i only needs a single piece of information to conclude hypothesis h_4 and h_5, whenever rule r_j succeeds, rule r_i also succeeds.

Definition 6.3: Given a **KB** = (**E, H, R**), if there is a pair of rules, r_i, r_j $\in$ **R** and i $\neq$ j, such that

(1) LHS(r_i) $\subset$ LHS(r_j) and

(2) RHS(r_i) = RHS(r_j)

then the rule r_i *subsumes* rule r_j, or say, rule r_j is subsumed by rule r_i.

Case II: One rule contains fewer constraints on the situations in which it will succeed and this rule can get more conclusions comparing with other rule.

Example 6.4: Given a **KB** = (**E, H, R**), and r_i, r_j $\in$ **R**, i $\neq$ j and

$r_i : e_1 \rightarrow h_4 \wedge h_5$

$r_j : e_1 \wedge h_1 \rightarrow h_5$.

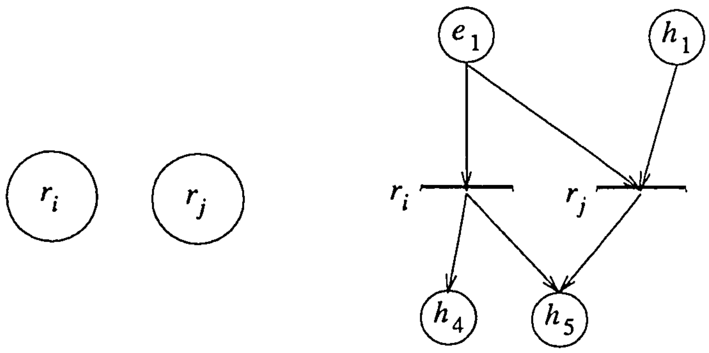

Figure 6.4 RDG and **KBNet** for example 6.4.

Definition 6.4: Given a **KB** = (**E, H, R**), if there is a pair of rules, r_i, r_j $\in$ **R** and i $\neq$ j, such that

(1) LHS(r_i) $\subseteq$ LHS(r_j) and

(2) RHS(r_j) $\subset$ RHS(r_i)

then the rule r_i *subsumes* rule r_j, or say, rule r_j is *subsumed* by rule r_i.

6.2.4. Redundant triangle

Considering another situation :

Example 6.5: Given a **KB** = (**E, H, R**), and rules $r_i, r_j, r_k \in$ **R**, they are :

$r_i : h_1 \rightarrow h_2$

$r_j : h_2 \rightarrow h_3$

$r_k : h_3 \rightarrow h_1$

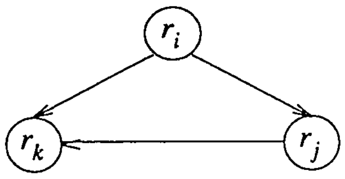

Figure 6.5 RDG for simple redundant triangle.

The rules rules r_i, r_j and r_k construct a *triangle*, the generalized situation is shown in following:

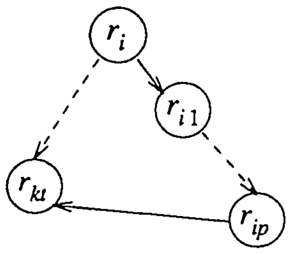

Figure 6.6 RDG for generalized redundant triangle.

Note here the dashed arrow does not mean partial dependency (represented by dotted arrow), it indicates that there are several rules with totally dependent relation along this direction. We get the following definition :

Definition 6.5: Given a **KB** = (**E, H, R**), and subset of rules, { $r_i, r_{i1}, r_{i2}, ..., r_{ip}, r_{k1}, r_{k2}, ..., r_{kt}$ } $\subseteq$ **R**, such that

(1) $r_i \propto r_{k1} \propto r_{k2} ... \propto r_{kt}$, $t \geq 1$;

(2) $r_i \propto r_{i1} \propto r_{i2} \propto ... \propto r_{ip}$, $p \geq 1$;

(3) $r_{ip} \propto r_{kt}$.

let RT = { $r_i, r_{i1}, r_{i2}, ..., r_{ip}, r_{k1}, r_{k2}, ..., r_{kt}$ }, the subset of rules, RT is called a *redundant triangle.*

Notice that the redundant triangle is different from a strong cycle, which is in below (the details will be seen in Chapter 7). The **RDG** in Figure 6.7 is a *strong cycle*, and the **RDG** in Figure 6.8 is a simple redundant triangle.

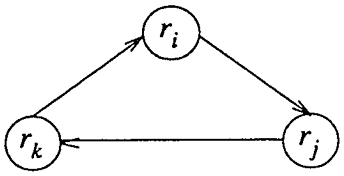

Figure 6.7 RDG for a strong cycle.

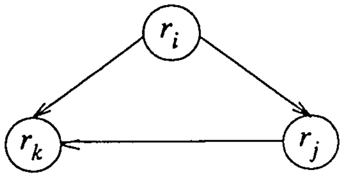

Figure 6.8 RDG for a redundant triangle.

The definition for strong cycle will be given in the next chapter. However, by looking at Figures 6.7 and 6.8, we can easily find the difference between the redundant triangle and the strong cycle. For the redundant triangle, we have:

T_closure(r_i) = { r_i, r_j, r_k, r_k }, and

T_closure(r_j) = { r_j, r_k }, and

T_closure(r_k) = { r_k }.

We have #(r_i, T_closure(r_i)) = 2, #(r_j, T_closure(r_j)) = 1, and #(r_k, T_closure(r_k)) = 1. But, for the strong cycle, we have

T_closure(r_i) = { r_i, r_j, r_k, r_i }, and

T_closure(r_j) = { r_j, r_k, r_i, r_j }, and

T_closure(r_k) = { r_k, r_i, r_j, r_k }.

We have #(r_i, T_closure(r_i)) = #(r_j, T_closure(r_j)) = #(r_k, T_closure(r_k)) = 2.

In terms of **KB** model and definitions for redundant rules, we offer the following definition:

Definition 6.6: A **KB** is minimal if it does not contain any kinds of following rules:

(1) Self redundant rules;

(2) Redundant rules;

(3) Redundant subsumed rules;

(4) Redundant triangle rules.

6.3. Algorithms for Checking Redundancy

In this section we give a set of algorithms which are used for checking redundancy of a **KB** - different redundant rules. The solutions are polynomial (in running time) in the number of rules, hence the knowledge elements of the rule set **R** of the **KB**. Included are algorithms for checking self redundant rules, redundant rules, subsumed redundant rules and redundant triangles.

6.3.1. Algorithm 6.1 for Checking Self Redundant Rule

Let **KB** = (**E, H, R**) be a knowledge base and $\mathbf{R} = \mathbf{R}_{\mathrm{I}} \cup \mathbf{R}_{\mathrm{II}} \cup \mathbf{R}_{\mathrm{III}}$. As a first step for checking redundancy of the **KB**, we give algorithm 6.1 to find self redundant rules.

```
Algorithm 6.1 Checking self-redundant rule
Input : Rule set $\mathbf{R} = \mathbf{R}_{\mathrm{I}} \cup \mathbf{R}_{\mathrm{II}} \cup \mathbf{R}_{\mathrm{III}}$
Output: Information about self redundant rule
Procedure:
  1. Arbitrary arrange of the rules, $r_1, ..., r_k$.
  2. For i := 1 to k do
       if LHS($r_i$) $\cap$ RHS($r_i$) $\neq \emptyset$
        then
         Self_redundant($r_i$) $\leftarrow$ true.
  3. end.
```

The correctness of the algorithm 6.3 is obvious. The complexity of the algorithm 6.1 is O(|**R**| × L), or O(K × L), here the L is the average size of rule r, r ∈ **R**. So we have the theorem 6.1.

Theorem 6.1: The algorithm 6.1 checks self redundant rules in time O(K × L).

6.3.2. Algorithm for Checking Redundant Rules

For a given **KB** =(**E, H, R**), if one of a pair of rules r_i, r_j is redundant, then we have LHS(r_i) = LHS(r_j) and RHS(r_i) = RHS(r_j). We can partition all the rules in **R** according to the size of the left hand side of each rule. We can divide the set of rules into q groups, $G_1, G_2, ..., G_i, ..., G_q$. The *i* of G_i represents that the size of "IF part" of all rules in group G_i is i. We can check the redundant rule in each group of rules, G_i, $q \geq i \geq 1$.

```
Algorithm 6.2 Check Redundant Rules
Input : Groups of production rules, $G_1, G_2, ..., G_q$,
        and $\mathbf{R} = G_1 \cup G_2 \cup ... \cup G_q$.
Output: Information about redundant rules
Procedure:

  1. Let $S = \{ G_1, G_2, ..., G_q \}$
  2. for i = 1 to q do
     2.1 pick a rule $r_x$ from $G_i$
     2.2 $G_i \leftarrow G_i - r_x$
     2.3   for each rule $r_y$ from $G_i$ do
     2.4     if LHS( $r_x$ ) = LHS( $r_y$ ) and
               RHS( $r_x$ ) = RHS( $r_y$ )
             then
             Redundant_rule($G_i$, $r_x$, $r_y$) $\leftarrow$ true
  3. end.
```

It is easy to prove the correctness of this algorithm. We discuss its complexity. Notice that we arrange the rules in an arbitrary order $r_1, r_2, ..., r_s$, in group G_t, $1 \leq t \leq q$. If we assume that the total number of rules in rule set **R** is K, $|\mathbf{R}| = K$, and the average number of rules in the group G_i, $1 \leq i \leq q$, is h, K/q = h, then the complexity of the algorithm 6.2 in the worst case is :

$\frac{1}{2} \times h \times (h - 1) \times q$

$= \frac{1}{2} h \times h \times (h - 1) \times K / h$

$= \frac{1}{2} K \times q \times (h-1)$

$\approx \frac{1}{2} K \times h$

$= O(K^2/2q)$.

So, we can get the following theorem:

Theorem 6.2 Algorithm 6.2 checks redundant rules in time $O(K^2/2q)$.

6.3.3. Algorithm for Checking Subsumed Rules

Given a set of rules **R**, and $\mathbf{R} = G_1 \cup G_2 \cup ... \cup G_q$, if rule r_i subsumes rule r_j, we should have

1. $r_i \in G_i$, and $r_j \in G_j$
2. $i \leq j$.

Based on this idea, we design the following algorithm for checking subsumed redundant rules.

Algorithm 6.3 Check Subsumed Rules

Input : R_I, groups of production rules, $G_1, G_2, ..., G_q$.

Output: Information about subsumed rules

Procedure:

```
1.  Let S = { $G_1, G_2, ..., G_q$ }
2.  for i = 1 to q do
    2.1  pick a rule $r_x$ from $G_i$
    2.2  $G_i \leftarrow G_i - r_x$
    2.3    for each rule $r_y$ from $G_i$ do
    2.4      if LHS( $r_x$ ) $\subseteq$ LHS( $r_y$ ) and
             RHS( $r_y$ ) $\subset$ RHS( $r_x$ )
```

```
                then
                  subsumed_rule1($G_i$, $r_x$, $r_y$) $\leftarrow$ true
3.        if i < q then
  3.1       for j := i+1 to q do
  3.2         for each rule $r_y$ from $G_j$ do
              if LHS( $r_x$ ) $\subseteq$ LHS( $r_y$ ) and
                 RHS( $r_y$ ) $\subset$ RHS( $r_x$ )
               then
                 subsumed_rule2($G_i$, $G_j$, $r_x$, $r_y$) $\leftarrow$ true
4.   end.
```

Theorem 6.3 Algorithm 6.3 checks the subsumed rules in time O(K^2/q).

Proof: If we assume that the total number of rules in rule set **R** is K, |**R**| = K, and the average number of rules in the group G_i, q ≥ i ≥ 1, is h, K/q = h, then the analysis is as follows:

(1) Step 2: inside the loop 'for i = 1 to q do', from step 2.1 to step 2.4, we have:

$$\begin{aligned}
&\tfrac{1}{2} \times h \times (h - 1) \times q \\
&= \tfrac{1}{2} h \times h \times (h - 1) \times K / h \\
&= \tfrac{1}{2} K \times q \times (h-1) \\
&\approx \tfrac{1}{2} K \times h \\
&= O(\tfrac{1}{2} K^2/q).
\end{aligned}$$

(2) Step 3: from step 3 to 3.2,

since $h_1 = h_2 =, ..., = s_q$ = h, we have:

$$\begin{aligned}
&\sum h_i \times h_j \times L \text{ (here, i from 1 to q-1, and j=i+1)} \\
&= h^2 \times (q-1) \\
&= O\,(h^2 \times q) \\
&= O\,(K^2/q)
\end{aligned}$$

So the complexity of Algorithm 6.3 is :

$$O(\ K^2/2q + h^2 \times q\ )$$
$$= O(\ K^2/2q + K^2/q\ )$$
$$\approx O(K^2/q\ )$$

□

6.3.4. Algorithm for Checking Redundant Triangle Rules

There are two basic cases for the redundant triangle, according to the definition 6.5. Assume that we have a redundant triangle,

$$RT = \{\ r_i, r_{i1}, r_{i2}, \ldots, r_{ip}, r_{k1}, r_{k2}, \ldots, r_{kt}\ \}.$$

Case I: rule $r_i \in \mathbf{R}_{III}$, this implies RT $\subseteq \mathbf{R}_{III}$, that is , all rules in the redundant triangle belong to type III rules.

Case II: rule $r_i \in \mathbf{R}_I \cup \mathbf{R}_{II}$, and RT - $\{r_i\} \subseteq \mathbf{R}_{III}$. The rule r_i has at least two total successor rules, rule r_{i1} and rule r_{k1}.

Since we have eliminated all self redundant rules, redundant rules and subsummed rules by using previous algorithms 6.1, 6.2 and 6.3, none of the rules in the RT are redundant or subsummed. The following algorithm 6.4 can be used either for checking redundant triangle in Case 1, if we only give the Type III rules ($\mathbf{R}_{III}$) as input or for checking redundant triangle in Case 2, if we give the all rules as input for this algorithm.

Algorithm 6.4 Check Redundant Triangle

```
Input : Type III rules ($\mathbf{R}_{\mathrm{III}}$) and r $\in$ $\mathbf{R}_{\mathrm{III}}$ (Case 1)
        or rule r $\in$ $\mathbf{R}_{\mathrm{I}} \cup \mathbf{R}_{\mathrm{II}}$ (Case 2)
Output: Information about redundant triangle rules
Procedure : ( depth first search for selecting rule at step 4 )
    1. For each rule r $\in$ $\mathbf{R}_{\mathrm{I}}$ or r $\in$ $\mathbf{R}_{\mathrm{I}} \cup \mathbf{R}_{\mathrm{II}}$ do
    2. T_closure(r) $\leftarrow$ {r}
    3. While T_closure(r) contain unchecked rules do
    4.     select an unchecked rule $r_k$ from T_closure(r) and mark it
    5.     for each rule $r_i \in \mathbf{R}_{\mathrm{III}}, r_i \neq r_k$
               5.1 if $|\mathrm{LHS}(r_i)| \leq |\mathrm{RHS}(r_k)|$ and $\mathrm{LHS}(r_i) \subseteq \mathrm{RHS}(r_k)$
                   then
               5.2   if $r_i \in$ T_closure(r)
                     and
               5.3   $\nexists\, r_x \in$ T_closure(r) such that $\mathrm{LHS}(r_x) \subseteq \mathrm{RHS}(r_i)$
                     then report redundant triangle rules, exit
                     else T_closure(r) $\leftarrow$ T_closure(r) + $\{r_i\}$
    6. stop.
```

We prove the correctness of the algorithm in the following theorem:

Theorem 6.4 Algorithm 6.4 checks out the redundant triangle correctly.

Proof: We have to discuss two cases for checking redundant triangles: in case I, the rule r $\in$ $\mathbf{R}_{\mathrm{III}}$; in case II, the rule r $\in$ $\mathbf{R}_{\mathrm{I}}$ or r $\in$ $\mathbf{R}_{\mathrm{II}}$.

Case I: If the rule set $\mathbf{R}_{\mathrm{III}}$ does not contain any redundant triangle rules beginning with rule r, then after finite steps the algorithm will stop and without reporting redundant triangle rules. Otherwise, the steps from 3 to 5.2 will check out the existing redundant triangle rules.

Case II: If there is a triangle starting with rule r, then there are two rules r_{k1} and r_{i1} which are total successors of the rule r, and $r_{k1}, r_{i1} \in$ **R_{III}**. The steps from 3 to 5.2 will check out such a redundant triangle. □

Considering the complexity of the algorithm 6.4, we have the following result:

Theorem 6.5 Algorithm 6.4 checks out the redundant triangle in time O(K_3^3) for Case 1 and ($(K_1 + K_2) \times K_3^2$) for Case 2.

Proof: We analyze Case 1 first:

Case 1: The rule r belongs to Type III rules, r $\in$ **R_{III}**.

Step 3 and step 4: After selecting an unchecked rule from T_closure(r), the T_closure(r) will be added with n_i unchecked rules, here n_i ≥ 0, p ≥ i ≥ 0, p ≥ 0 and $n_0 = 1$. So, at last the maximum size of T_closure(r) is :

$$|T\_closure(r)| = n_0 + n_1 +, ..., + n_p.$$

We have K_3 + 1 ≥ |T_closure(r)| ≥ 1.

Step 5 will cost O(K_3) time. For each rule checking, the cost is O(K_3^2).

So the complexity of algorithm 6.4 for Case I is O(K_3^3).

Case II: The rule r belongs to Type I or Type II rules. The rest of cost for the algorithm is the same as Case 1, so the complexity of algorithm 6.4 for Case 2 is O($(K_1 + K_2) \times K_3^2$).

6.4. Example

We will use the **BAGGER** example, introduced in Chapter 4, to illustrate some ideas presented in this chapter. Suppose that **BAGGER** system contains the following rules:

$r_1 : e_1 \wedge e_2 \rightarrow h_1$

$r_2 : e_3 \wedge e_4 \wedge h_2 \rightarrow h_2$

$r_3 : e_5 \wedge e_6 \rightarrow h_3$

$r_4 : e_1 \wedge e_7 \rightarrow h_4$

$r_5 : e_2 \wedge e_8 \rightarrow h_4$

$r_6 : e_3 \wedge e_9 \rightarrow h_4$

$r_7 : e_{10} \rightarrow h_5$

$r_8 : e_1 \wedge e_2 \rightarrow h_1$

$r_9 : h_4 \rightarrow h_3$

$r_{10} : h_3 \rightarrow h_2$

$r_{11} : h_4 \rightarrow h_2$

The self redundant rule is r_2, a redundant rule is r_8 and a redundant triangle is RT = { r_9, r_{10}, r_{11} }.

Summary

In this chapter we have used two models, **KBS** and **KB**, to discuss the minimality problem of a knowledge base system. In particular, we checked the redundant rules in terms of **KB** model. We have identified *self redundant rule, redundant rule, subsummed rule* and *redundant triangle rules.* A set of algorithms was given for checking such redundant rules. The correctness and complexity of the algorithms were also proved.

7.

CHAPTER 7

TERMINATION PROBLEM

The termination problem of a knowledge base system is another important issue in terms of our definition of effectiveness. Previous work on termination problem has either assumed that a rule set of a knowledge base was acyclic [17,31,32,85,86], or simply claimed that cycles of a rule set may lead to a nonterminating problem [50-52]. They either ignored the cycles in a knowledge base or the characteristics of the cycles in the rule set of the knowledge base. The former did not consider the termination problem of a knowledge base and the latter did not further analyze the sufficient and necessary conditions for the termination problem of the knowledge base. In this chapter, we use the general qualitative model **KB** for knowledge bases, **KBNet** and **RDG** (all of them are defined before), and systematically analyze the termination problem of a given knowledge base. The mechanism is implicit token propagation, and a rule is one time firing unless the evidences and hypothesis on its left hand side are given again. We identify three kinds of cycles: *strong apple cycle, semi strong apple cycle*, and *weak apple cycle*, which may exist in a given set of rules. They have different characteristics and give different influences for **KB** terminating. A set of algorithms is presented to recognize such cycles, and some theoretical results are also given. When we work on the termination problem of a knowledge base system, we have the following assumptions: (1) we discuss the terminating problem under the **KB** model; (2) we assume that the redundant rules were eliminated; the **KB** does not contain the self redundant, redundant, subsummed redundant and redundant triangle rules (see previous chapter).

7.1. Problem Formulation

7.1.1. Termination and Cycles

Given a knowledge base system **KB**, its **KBNet** and **RDG**, let us to consider its termination problem. If the **RDG** contains some cycles, the cycles may cause nontermination of the knowledge base systems. Suppose a given **RDG** looks like that in Figure 7.1:

Example 7.1 Given a rule set **R** = $\{ r_1, r_2, r_3, r_4, r_5 \}$, its **RDG** is the following :

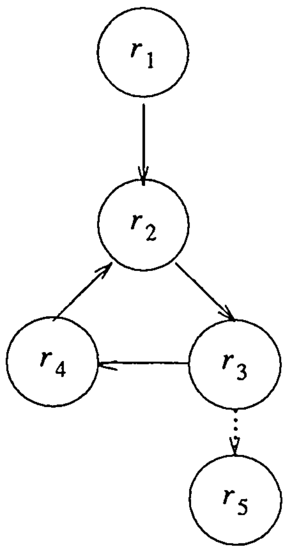

Figure 7.1 RDG for Example 7.1

The situation in Figure 7.1 implies that execution of this knowledge base system may lead to an infinite loop; e.g., the firing sequence is $\Pi = \{ r_1, r_2, r_3, r_4, r_2, r_3, r_4, \ldots \}$, it can be formulated as :

$\Pi = \{ r_1, (r_2, r_3, r_4)^N \}$, N is natural number, and $N \rightarrow \infty$.

This property is not desirable because it indicates the nontermination condition of the **KB** according to the implicit token propagation mechanism.

Looking at the problem from another aspect, it is slightly different. Recalling that in a rule dependency graph, edge(r_i, r_j) $\in$ $\mathbf{E_T}$ or edge(r_i, r_j) $\in$ $\mathbf{E_P}$ if and only if $r_i \propto r_j$ or $r_i < r_j$ holds respectively. The relation '<' is partial dependency, and it is not total dependency. This indicates that if the $r_i < r_j$ holds and rule r_i is fired, the rule r_j may or may not be fired.

Example 7.2 (Continue) Given a rule set **R** = { r_1, r_2, r_3, r_4, r_5 } and suppose that the rules are :

$$r_1 : e_1 \wedge e_2 \rightarrow h_1$$
$$r_2 : e_3 \wedge h_1 \rightarrow h_2 \wedge h_3$$
$$r_3 : e_4 \wedge h_2 \rightarrow h_4 \wedge h_5$$
$$r_4 : e_5 \wedge h_4 \rightarrow h_1$$
$$r_5 : e_6 \wedge h_5 \rightarrow h_6$$

Its **RDG** is :

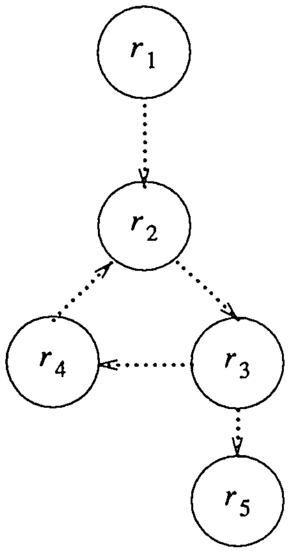

Figure 7.2 The RDG for Example 7.2

The corresponding **KBNet** is as follows:

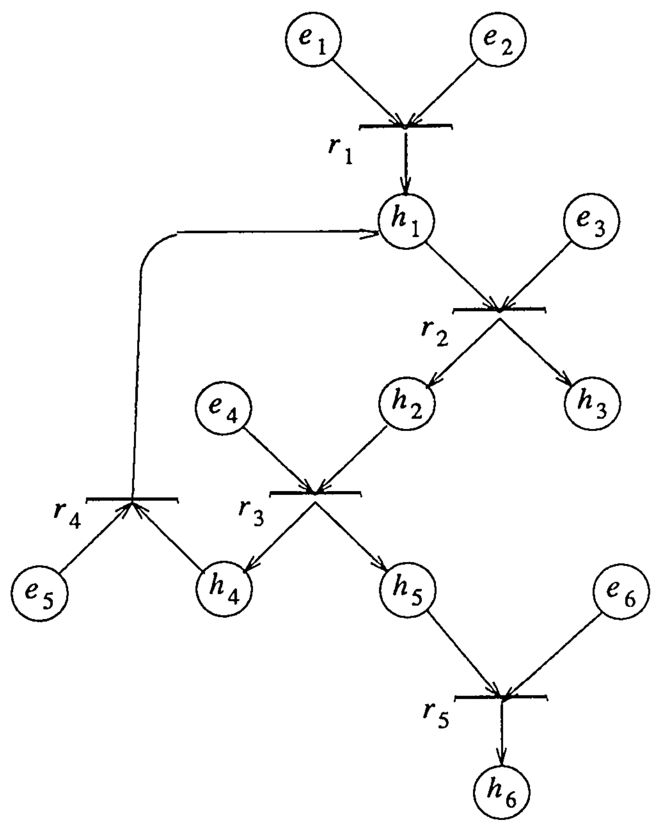

Figure 7.3 KBNet for Example 7.2.

The *possible firing sequence* is $\Pi = \{ r_1, r_2, r_3, (r_4,r_5), r_2, r_3, (r_4,r_5), \ldots \}$; the cycled firing sequence exists if and only if the all evidences, $\{ e_1, e_2, e_3, e_4, e_5 \}$, are true. If one of them is not true, the cycle in the **KBNet** can not cause nontermination. Whether all the evidences, $\{ e_1, e_2, e_3, e_4, e_5 \}$, can be true simultaneously belongs to application domain semantic constraints [97].

From examples 7.1 and 7.2, we noticed that the cycles in the rule dependency graph may cause the nontermination of a knowledge base. The key issue for the termination of a knowledge base is to determine whether its rule set contains any cycles and to analyze their characteristics. There are three different kinds of cycles within the rule set of a knowledge base. Their characteristics are described in the following.

7.1.2. Strong Apple Cycle

let us check an example first.

Example 7.3 Given a rule set **R** = { r_1, r_2, r_3, r_4 } and the rules are :

$$r_1 : e_1 \rightarrow h_1$$

$$r_2 : h_1 \rightarrow h_3$$

$$r_3 : h_3 \rightarrow h_4$$

$$r_4 : h_4 \rightarrow h_1$$

Notice that $r_1 \in \mathbf{R_I}$ (Type I), { r_2, r_3, r_4 } $\subset \mathbf{R_{III}}$ (Type III). The corresponding **KBNet** and **RDG** are :

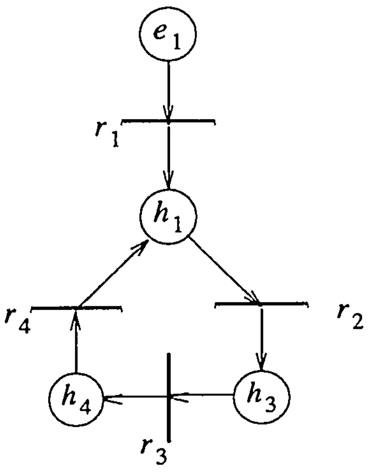

Figure 7.4 KBNet for Example 7.3.

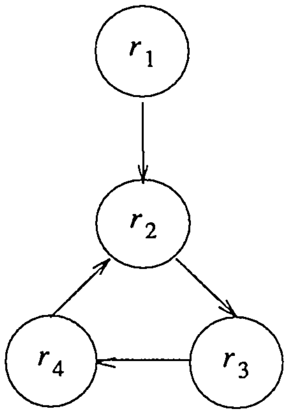

Figure 7.5 RDG for strong apple cycle in example 7.3

Such cycles have some special characteristics. First, all rules except rule r_1 belong to Type III (**H** → **H**). Second, the set of rules has the totally dependent relation : $r_1 \propto r_2$, $r_2 \propto r_3$, $r_3 \propto r_4$, and $r_4 \propto r_2$. Third, T_closure(r_1) = { r_1, r_2, r_3, r_4, r_2 } (recalling, we have said that the closure is a restricted bag). Fourth, such cycle forms as an *apple* kind of cycle in general:

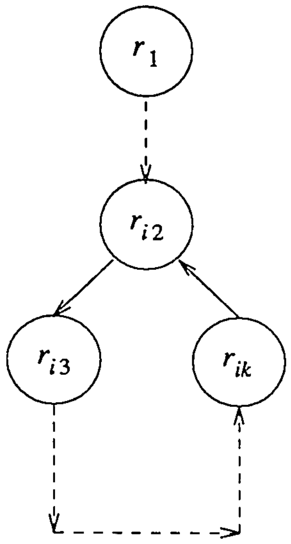

Figure 7.6 RDG for general strong apple cycle

Notice that it is possible to have other rule(s) from rule r_1 to r_{i2}, and also possible to have other rule(s) between rule r_{i3} and r_{ik}. We use the dashed arrow in Figure 7.6 to represent this situation. From a pragmatic point of view, if the rule r_1 is never firable, then this set of rules { $r_{i2}, r_{i3}, ..., r_{ik}, ...$ } is not usable, or say, the set of rules constructs an isolated cycle. These characteristics lead to following general definition:

Definition 7.1 A subset of rules S, S ⊆ **R**, S = { $r_1, r_2, ..., r_p$ }, p ≥ 3, if the relation $r_1 \propto r_2$, $r_2 \propto r_3$, ..., $r_{p-1} \propto r_p$, and $r_p \propto r_q$ hold, $2 \le q \le$ p - 1, then the S is a *strong apple cycle*, and the firing sequence Π_{stem} = { $r_1, ..., r_{q-1}$ } is called *stem sequence.*

Definition 7.2 A subset of rules S, S ⊆ $\mathbf{R}_{\mathrm{III}}$, S = { $r_{i1}, r_{i2}, ..., r_{ip}$ }, p ≥ 2, if the relation $r_{i1} \propto r_{i2}$, $r_{i2} \propto r_{i3}$, ..., $r_{ip-1} \propto r_{ip}$, and $r_{ip} \propto r_{i1}$ holds, then the S is a *strong cycle.*

From the definitions above we can get some immediate results:

Lemma 7.1 For a rule $r \in \mathbf{R}_{III}$, if

(1) #(r, T_closure(r)) = 2, or

(2) #(r_i, T_closure(r)) = 2 and $r \neq r_i$

then there is a strong cycle.

Proof: If #(r, T_closure(r)) = 2, then according to the definition of T_closure(r), there is a subset of rules $S \subseteq \mathbf{R}_{III}$, $S = \{ r_{i1}, r_{i2}, ..., r_{ip} \}$, $p \geq 2$, and T_closure(r) = S.

(1) If |T_closure(r))| = 2, we have :

$$r \propto r \propto r$$

this is a self strong cycle.

(2) If |T_closure(r))| > 2, we have :

$$r = r_{i1} \propto r_{i2} \propto r_{i3}, ..., \propto r_{i,p-1} \propto r_p = r,$$

this is a strong cycle.

(3) If |T_closure(r))| > 2, #(r_i, T_closure(r)) = 2 and $r \neq r_i$, there is a subset of rules $S \subseteq \mathbf{R}_{III}$, $S = \{ r_{i1}, r_{i2}, ..., r_{ip} \}$, $p > 2$, and T_closure(r) = S, such that

$$r_{ij} = r_i, 2 \leq j \leq p\text{-}1, \text{ and } r_{ij} = r_i$$

the relation

$$r = r_{i1} \propto r_{i2} \propto r_{i3}, .. \propto r_{ip} \propto .., ..., r_{i,p-1} \propto r_p = r_i \text{ hold,}$$

the rule firing sequence

$$r_i = r_{ip} \propto .., ..., r_{i,p-1} \propto r_p = r_i$$

is a strong cycle. □

Lemma 7.2 If a **KB** rule set **R** contains some strong apple cycles, and the related stem sequence is firable, then the **KB** is nonterminating.

Proof: Suppose not. The **KB** contains a strong apple cycle, and the stem sequence is Π_k. After the Π_k is fired, the strong cycle will lead to an infinite loop and the **KB** will never stop its execution. Contradiction. □

7.1.3. Semi Strong Apple Cycle

Let us consider another situation. Look at Example 7.4 first:

Example 7.4 Given a rule set **R** = { $r_1, r_2, r_3, r_4, r_5, r_6$ } and the rules are :

$r_1 : e_1 \rightarrow h_1$	$r_4 : h_2 \rightarrow h_5$
$r_2 : h_1 \rightarrow h_2 \wedge h_3$	$r_5 : h_4 \wedge h_5 \rightarrow h_6$
$r_3 : h_3 \rightarrow h_4$	$r_6 : h_6 \rightarrow h_3$

Notice that $r_1 \in \mathbf{R_I}$ (Type I), { r_2, r_3, r_4, r_5, r_6 } $\subset \mathbf{R_{III}}$ (Type III), and $r_4 < r_5, r_3 < r_5$.

The corresponding **KBNet** and **RDG** are :

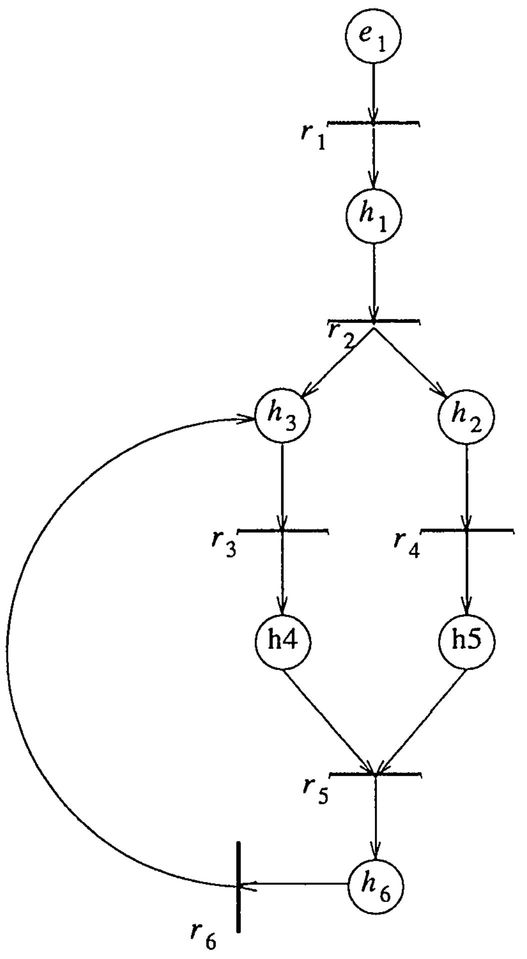

Figure 7.7 **KBNet** for Example 7.4.

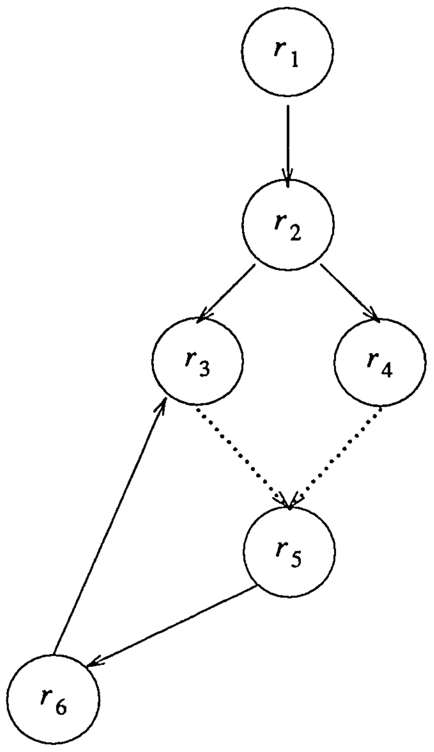

Figure 7.8 RDG for semi strong apple cycle in example 7.4.

Such a cycle has some similar characteristics with the strong apple cycle: except for rule r_1, all rules belong to Type III (**H** $\rightarrow$ **H**); ST_closure(r_1) = { $r_1, r_2, r_3, r_4, r_5, r_6, r_3$ }, some rule, like r_3, has two occurrences in the ST_closure(r_1).

But, such a cycle has some special characteristics which are different from the strong apple cycle. First, the chain does not satisfy the totally dependent relation; instead, the chain reflects the semi totally dependent relation : $r_1 \propto r_2$, $r_2 \propto \{r_3,r_4\}$, $\{r_3,r_4\} \propto r_5$, ($r_3 < r_5$ and $r_4 < r_5$), $r_5 \propto r_6$, and $r_5 \propto r_6$, the T_closure(r_1)= { r_1, r_2, r_3, r_4 } which does not implicate any cycle. Second, the ST_closure(r_1) forms a semi apple cycle.

This situation has another variate: if we have the rule r_6 : $h_6 \rightarrow h_1$, the **RDG** has another form:

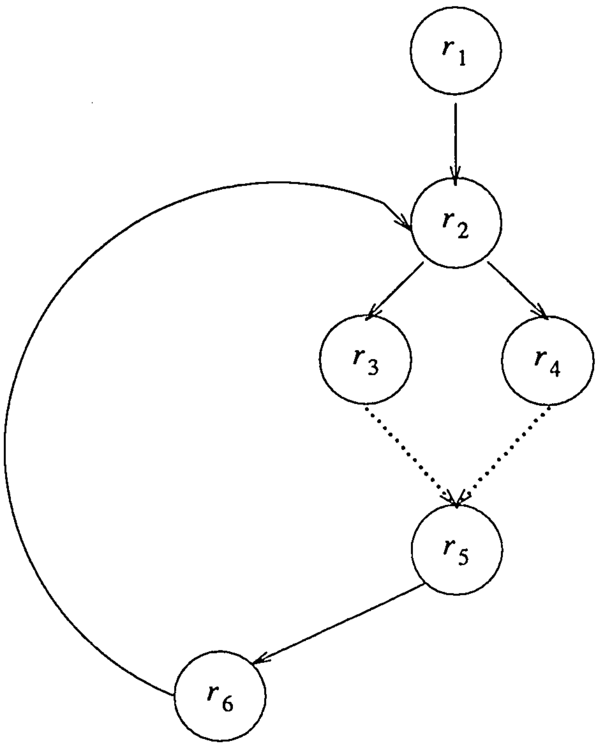

Figure 7.9 RDG for modified example 7.4.

The general case can be described as follows:

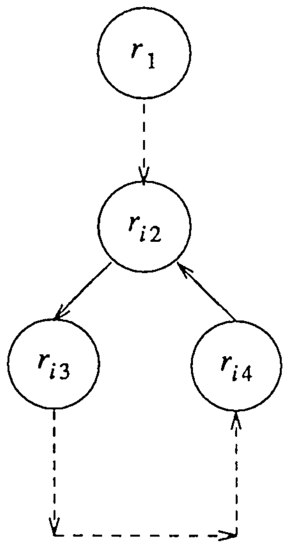

Figure 7.10 RDG for general semi strong apple cycle.

It is possible to have other rule(s) between rule r_1 and rule r_{i2}, and between rule r_{i3} and r_{i4}. The dashed arrow in Figure 7.10 is used for describing these cases. The properties of the apple cycle still remain: if the rule r_1 (stem) is never firable, then this set of rules { $r_2, r_3, r_4, \ldots$ } is useless. These characteristics lead to other definitions:

Definition 7.3: A subset of rules S, $S \subseteq \mathbf{R}$, $S = \{ r_1, r_2, \ldots, r_p \}$, $p \geq 3$, let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i \leq p$, if the relation $r_1 \propto \beta_1, \beta_1 \propto \beta_2, \ldots, \beta_t \propto \delta_q$, $2 \leq t \leq p-1$ and $2 \leq q \leq t$, here $\delta_q \in \beta_q$ or $\delta_q \subseteq \beta_q$, then the S is a *semi strong apple cycle*, and rule firing sequence $\Pi_{\text{stem}} = \{ r_1, \beta_1, \ldots, \beta_{q-1} \}$ is called *stem sequence.*

Definition 7.4:A subset of rules S, $S \subseteq \mathbf{R}$, $S = \{ r_1, r_2, \ldots, r_p \}$, $p \geq 3$, let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i < p$, if the relation $\beta_1 \propto \beta_2, \beta_2 \propto \beta_3, \ldots, \beta_q \propto \delta_1$, $2 \leq q \leq p-1$, here $\delta_1 \in \beta_1$ or $\delta_1 \subseteq \beta_1$, then the S is a *semi strong cycle.*

From the definitions, we have the following results:

Lemma 7.3 For a rule $r \in \mathbf{R}_{\text{III}}$, ST_closure(r) = $\{ r_1, r_2, \ldots, r_p \}$, if

(1) #(r, ST_closure(r)) = 2 or

(2) #(r_i, ST_closure(r)) = 2, and $r_i \neq r$,

then there is a semi strong cycle.

Proof: We prove the lemma through following two cases :

(1) If #(r, ST_closure(r)) = 2, then according to the definition of ST_closure(r), there is a subset of rules $S \subseteq \mathbf{R}_{\text{III}}$, $S = \{ r_{i1}, r_{i2}, \ldots, r_{ip} \}$, $p \geq 3$, and ST_closure(r) = S. let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i < p$, we have :

$$r = r_{i1} \propto \beta_1 \propto \beta_2 \propto, \ldots, \propto \beta_t \propto r_{i1} = r,\ 1 \leq t < p$$

this is a semi strong cycle.

(2) If #(r_i, ST_closure(r)) = 2 and $r_i \neq r$, there is a rule firing sequence

$$r = r_{i1} \propto \beta_1 \propto \beta_2 \propto, \ldots \propto \beta_{iq}, \ldots, \propto \beta_t \propto \delta_q = r_i,\ 1 \leq t < p \text{ and}$$
$$1 < q \leq t,\ \delta_q \in \beta_q \text{ or } \delta_q \subseteq \beta_q,$$

the sequence

$$\beta_q \propto \beta_{q+1} \propto, ..., \beta_t \propto \delta_q = r_i, 1 \leq t < p \text{ and } 1 < q \leq t, \delta_q \in \beta_q \text{ or } \delta_q \subseteq \beta_q,$$

is a semi strong cycle. □

Lemma 7.4 If a rule set **R** of a **KB** contains some semi strong apple cycles, and the related stem rule(s) are firable, then the **KB** is potentially nonterminating.

Proof: Suppose not. The **KB** contains a semi strong apple cycle, and the stem sequence is Π_k. After the stem sequence is Π_k is fired, the semi strong cycle will lead to an infinite loop and the **KB** will never stop its execution. Contradiction. □

7.1.4. Weak Apple Cycle

Let us check another example :

Example 7.5 Given a rule set **R** = { r_1, r_2, r_3, r_4 } and the rules are :

$$r_1 : e_1 \wedge e_2 \rightarrow h_1$$

$$r_2 : h_1 \rightarrow h_2 \wedge h_3$$

$$r_3 : e_3 \wedge h_2 \rightarrow h_4 \wedge h_5$$

$$r_4 : h_4 \rightarrow h_1$$

Notice that $r_1 \in \mathbf{R}_\mathrm{I}$ (Type I), { r_2, r_4 } $\subset \mathbf{R}_\mathrm{III}$ (Type III), and $r_3 \in \mathbf{R}_\mathrm{II}$ (Type II). The corresponding **RDG** and **KBNet** are :

The **RDG** for this example is :

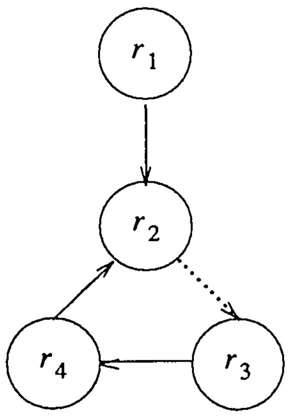

Figure 7.11 RDG for weak apple cycle in example 7.5.

The **KBNet** for it is :

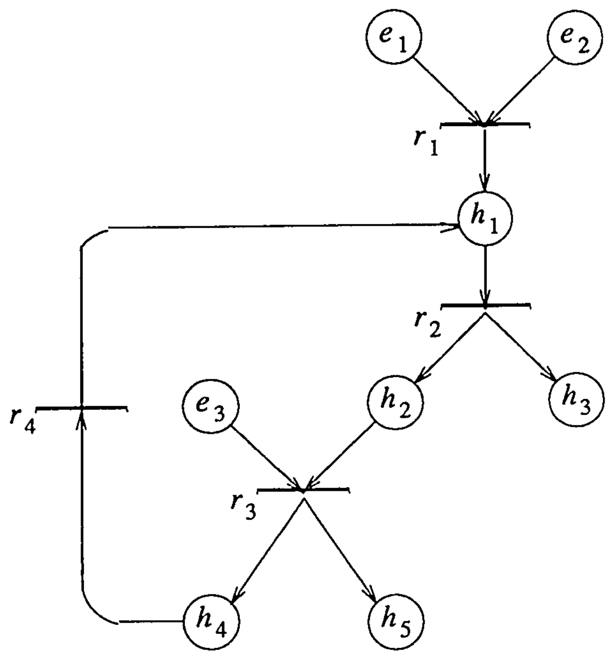

Figure 7.12 KBNet for weak apple cycle in example 7.5.

Compared with strong and semi strong apple cycles, this cycle has some special characteristics : some rule, like r_3, belongs to Type II; the dependent relation is : $r_1 \propto r_2$, $r_2 < r_3$, $r_3 \propto r_4$, and $r_4 \propto r_2$; T_closure(r_1) = { r_1, r_2 }, ST_closure(r_1) = { r_1, r_2 }, and P_closure(r_1) = { r_1, r_2, r_3, r_4, r_2 }. In other words, the relation between r_3 and r_4 is partially evidence dependent. This indicates that this kind of cycle may cause the **KB** to nonterminate, but not always. In our example, the **KB** is nonterminate if and only if the evidences { e_1, e_2, e_3 } are true. The evidential partial dependency only conditionally determines the termination of the **KB**. These lead us to define the following:

Definition 7.5: A subset of rules S, S $\subseteq$ **R**, S = { $r_1, r_2, ..., r_p$ }, p $\geq$ 3, let $\beta_i \in$ S or $\beta_i \subset$ S, 1 $\leq$ i < p, and the notation θ stands for "$\propto$" or "<" relation, that is $\theta \in$ { $\propto$, < }, if the relation

$$r_1 \,\theta\, \beta_1, \beta_1 \,\theta\, \beta_2, ..\theta\, \beta_q\, \theta, ..., \beta_t\, \theta\, \delta_q \text{ hold,}$$

here $2 \leq t \leq p-1$, $2 \leq q \leq t$, and $\delta_q \in \beta_q$ or $\delta_q \subseteq \beta_q$, then the S is a *weak apple cycle*, and the rule firing sequence $\Pi_{stem} = \{ r_1, \beta_1, ..., \beta_{q-1} \}$ is called *stem sequence.*

Notice that there is at least one occurrence of θ, which is "<" relation.

Definition 7.6: A subset of rules S, $S \subseteq \mathbf{R}$, $S = \{ r_1, r_2, ..., r_p \}$, $p \geq 3$, let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i < p$, and the notation θ stands for '∝' or '<' relation, if the relation

$$\beta_1 \theta \beta_2, \beta_2 \theta \beta_3, ..., \beta_q \theta \delta_1, 1 \leq q \leq p\text{-}1, \text{here } \delta_1 \in \beta_1 \text{ or } \delta_1 \subseteq \beta_1,$$

then the S is a *weak cycle.*

Lemma 7.5 For a rule $r \in (\mathbf{R}_{II} \cup \mathbf{R}_{III})$, if

(1) #(r, P_closure(r)) = 2, or

(2) #(r_i, P_closure(r)) = 2 and $r \neq r_i$

then there is a weak cycle.

Proof: We use the notation θ with the same meaning as above.

(1) If |P_closure(r)| = 2, it is trivial.

(2) If #(r, P_closure(r)) = 2, then according to the definition of ST_closure(r), there is a subset of rules $S \subseteq (\mathbf{R}_{III} \cup \mathbf{R}_{III})$, $S = \{ r_{i1}, r_{i2}, ..., r_{ip} \}$, $p \geq 2$, and P_closure(r) = S. let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i < p$, we have :

$$r = r_{i1} \theta \beta_1 \theta \beta_2 \theta.., ..., \theta \beta_t \theta r_{i1} = r, 1 \leq t < p$$

this is a weak cycle.

(3) If #(r_i, P_closure(r)) = 2 and $r \neq r_i$, then according to the definition of P_closure(r), there is a subset of rules $S \subseteq (\mathbf{R}_{III} \cup \mathbf{R}_{III})$, $S = \{ r_{i1}, r_{i2}, ..., r_{ip} \}$, $p \geq 2$, and P_closure(r) = S. let $\beta_i \in S$ or $\beta_i \subset S$, $1 \leq i < p$, we have :

$$r_{i1} \theta \beta_1 \theta \beta_2 \theta.., \theta \beta_q, ...,\beta_{t-1} \theta \beta_t \theta \delta_q, 1 \leq t < p$$

here $1 \leq t < p$, $1 \leq q < t$, and $\delta_q \in \beta_q$ or $\delta_q \subseteq \beta_q$.

The sequence

$$\beta_q \theta \beta_{q+1}, ..., \beta_{t-1} \theta \beta_t \theta \delta_q, 1 \leq t < p, \text{and } \delta_q \in \beta_q \text{ or } \delta_q \subseteq \beta_q,$$

is a weak cycle. □

Lemma 7.6 If a **KB** rule set **R** contains some weak apple cycles, and for a given set of input evidences,

(1) the related stem are firable, and

(2) each rule in the weak cycle is firable,

then the **KB** is nonterminating.

Proof: Suppose not. The **KB** contains a weak apple cycle, and the stem sequence is Π_k . After the Π_k is fired, since each rule in the weak cycle is firable, the weak cycle will lead in a infinite loop and the **KB** will never stop its execution. Contradiction. □

7.1.5. The Sufficient and Necessary Conditions for Termination Problem

Let us further define the notation "cycle":

Definition 7.7 The term *cycles* represents the strong cycle, semi strong cycle and weak cycle.

We can get the immediate results:

Theorem 7.1 Let K be a positive integer, the following three statements are equivalent:

(1) A **KB** is nonterminating;

(2) $\exists\, r_j, \forall$ K, $\exists\, \Pi$ in **KB** such that $\#(r_j, \Pi) > K$;

(3) There is an infinite firing sequence in **KB**, i.e., $\exists\, r_j, \exists\, \Pi$ such that $\forall$ K, $|\Pi| > K$;

Proof: From (1) to (2) and (3): Suppose a **KB** is nonterminating, then there is at least one firing sequence Π in **KB**, such that the length of Π is infinite. In another aspect, we can find a rule $r \in$ **R**, and a Π in **KB**, the number of occurrences of rule r in Π is infinite. The proofs of 'from (2) to (1) and (3)' and 'from (3) to (1) and (2)' are similar. We omit them here. □

Theorem 7.2 The sufficient condition for a **KB**'s termination is that it does not contain any cycles.

Proof: It is the direct result from **Lemma 7.1** to **7.6**.

Notice that the converse of **Theorem 7.2** is not true. We can easily find a counterexample to the converse. Suppose a **KBNet** contains a strong cycle, but we can not find its stem sequence. Because the rules of a such strong cycle can never be fired, such a cycle does not make the nontermination of the **KB**. This kind of cycle indicates that the rules of such strong cycle are useless.

Theorem 7.3 The sufficient and necessary condition for a **KB**'s termination is that it does not contain any firable cycles.

Proof:

(<=) If the **KB** does not contain any firable cycles, this condition will never lead to an infinite firing sequence, so, after finite steps of rule firings, the **KB** will be terminated.

(=>) If the **KB** terminates, then it does not contain any firable cycles. Suppose not. There is a firable cycle, which will lead to an infinite firing sequence Π. This is contradiction. □

7.2. Algorithms for Checking Cycles

In this subsection we develop some algorithms for checking a strong cycle, semi strong cycle and weak cycle for a given rule set of a knowledge base. These algorithms are based on the **KB** model and rule property defined in the previous chapters. We assume the existence and inexpensiveness of the operations: membership checking, intersection and subset checking. In this section, we will use the term *tractable* to refer that the order of computation is polynomial, and *intractable* to refer that the order of computation is exponential.

7.2.1. Algorithm for Checking Strong Cycle

Given a knowledge base, **KB** = (**E, H, R**), we design algorithm 7.1 for checking strong cycle. The input for the algorithm is rule set $\mathbf{R}_{III}$, since strong cycles exist within Type III rule set.

```
Algorithm 7.1 Check_a_strong_cycle(r)
Input : Type III rules, $\mathbf{R}_{III}$, and rule r $\in$ $\mathbf{R}_{III}$
Output: Information about strong cycle
Procedure :
 0. For each rule r $\in$ $\mathbf{R}_{III}$ do
 1. T_closure(r) $\leftarrow$ {r}
 2. While T_closure(r) contain unchecked rules do
 3.   Select an unchecked rule $r_k$ from T_closure(r) and mark it
 4.   For each rule $r_i \in \mathbf{R}_{III}, r_i \neq r_k$
 5.     If $|LHS(r_i)| \leq |RHS(r_k)|$ and $LHS(r_i) \subseteq RHS(r_k)$
        then
 6.       if $r_i \in$ T_closure(r)
           then report strong cycle, exit
           else T_closure(r) $\leftarrow$ T_closure(r) + $\{r_i\}$
```

7. Stop.

We prove the correctness of the algorithm in the following theorem:

Theorem 7.4 Algorithm 7.1 checks out the strong cycle correctly.

Proof: If the rule $r \in \mathbf{R}_{\mathrm{III}}$ is not contained in any strong cycles, this algorithm will be stopped at finite steps without reporting a strong cycle. If the rule is in some strong cycles, one of the strong cycles is correctly computed as indicated by the double loops of the lines 2 through 6. This is because of the follows:

(1) Line 4 and 5 will check all rules which have to be fired after rule r is fired;

(2) Line 6 will check all rules in rule set $\mathbf{R}_{\mathrm{III}}$ to find totally dependent ones;

(3) Line 3.2.3 decides whether there is a strong cycle. If we find a strong cycle, the algorithm will be stopped.

The theorem is proved. □

Theorem 7.5 Algorithm 7.1 checks out the strong cycle in time O(K_3^3).

Proof: If we just consider finding one strong cycle in a given rule set R_{III}, steps 2 and 3 take a sequence of iterations: 1, n_1, ..., n_t. This means that, in the first round, the T_closure(r) only contains 1 unchecked rule; in second round, the T_closure(r) contains n_1 unchecked rules, and so on. So, |T_closure(r)| = $1 + n_1 +, ... + n_t$. Step 4 takes K_3 times, here $|\mathbf{R}_{\mathrm{III}}| = K_3$, for each selected and unchecked rule. The complexity of Algorithm 7.1 is:

$$(|R_{III}|-1) \times 1 + (|R_{III}|-1) \times n_1 +, ..., + (|R_{III}|-1) \times n_t$$

$$= (|R_{III}|-1) \times (1 + n_1 +, ... + n_t) \times |R_{III}|$$

here $(1 + n_1 +, ... + n_t) = |\text{T\_closure(r)}|$

$$= (|R_{III}|-1) \times |\text{T\_closure(r)}| \times |R_{III}|$$

We have $1 \leq |\text{T\_closure(r)}| \leq |R_{III}|$, so we get the complexity in the worst case :

$$(|R_{III}|-1) \times (|R_{III}|-1) \times |R_{III}|$$

$$\leq (|R_{III}|-1) \times |R_{III}| \times |R_{III}|$$

$$\approx |R_{III}| \times |R_{III}| \times |R_{III}|$$
$$= O(|R_{III}|^3)$$
$$= O(K_3^3). \quad \square$$

Theorem 7.6 The problem of finding all strong cycles is intractable.

Proof: This problem can be reduced to the need to find all cycles in graph theory. We can view the rule set $\mathbf{R}_{\mathrm{III}}$ as a set of vertices, and the total dependency of a pair of rules '$\propto$' corresponds to direct arc. From graph theory, we have the following result: The total number of direct cycles in a digraph is exponential with the cardinality of the rule set $\mathbf{R}_{\mathrm{III}}$ [6]. This makes the problem to be intractable.

7.2.2. Algorithm for Checking Semi Strong Cycle

According to the definition of a semi strong cycle, such cycles exist in rule set $\mathbf{R}_{\mathrm{III}}$. The algorithm 7.2 is developed for checking semi strong cycles.

```
Algorithm 7.2 Check_a_semi_strong_cycle
Input : Type III rules, $\mathbf{R}_{\mathrm{III}}$, and rule r $\in$ $\mathbf{R}_{\mathrm{III}}$
Output: Information about a semi strong cycle
Procedure :
  1. For each rule r $\in$ $\mathbf{R}_{\mathrm{III}}$ do
  2. ST_closure(r) $\leftarrow$ {r}
  3. P $\leftarrow$ RHS(r)
  4. While ST_closure(r) contain unchecked rules do
  5.   Select an unchecked rule $r_k$ from ST_closure(r) and mark it
  6.   For each rule $r_i \in \mathbf{R}_{\mathrm{III}}$, $r_i \neq r_k$ do
  7.     If |LHS($r_i$)| $\subseteq$ P
         then
  8.       if $r_i \in$ ST_closure(r)
           then report semi strong cycle and exit
```

```
        else
9.        ST_closure(r) $\leftarrow$ ST_closure(r) $\cup$ $\{r_i\}$
10.       P $\leftarrow$ P $\cup$ RHS($r_i$)
11. Stop.
```

Theorem 7.7 Algorithm 7.2 checks out the semi strong cycle correctly.

Proof: If the rule r$\in \mathbf{R}_{III}$ is not contained in any semi strong cycles, this algorithm will be stopped at finite steps without reporting a semi strong cycle. If the rule r is in some semi strong cycles, one of the semi strong cycles is correctly computed as indicated by the double loops of lines 2 through 6. This is because of the follows:

(1) Lines 4 and 5 will check all rules which have to be fired after rule r is fired;

(2) Lines 6 and 7 will check all rules in rule set $\mathbf{R}_{III}$ to find totally dependent ones;

(3) Line 8 decides whether there is a semi strong cycle. If we find a semi strong cycle, the algorithm will be stopped.

The theorem is proved. □

Theorem 7.8 Algorithm 7.3 checks out a semi strong cycle in time O($K_3 \times$ L).

Proof:

Step 5 and step 6 cost ($|\mathbf{R}_{III}|$ - 1), or (K - 1).

Step 7 costs L, L is the average size of left hand side of rule r $\in \mathbf{R}_{III}$.

Step 7 costs O(1).

So the complexity is:

$$
\begin{aligned}
&(|\mathbf{R}_{III}| - 1) \times (L + 1) \\
&\approx |\mathbf{R}_{III}| \times L \\
&= O(|\mathbf{R}_{III}| \times L) \\
&= O(K_3 \times L) \; \square.
\end{aligned}
$$

Theorem 7.9 The problem of finding a semi strong cycle is intractable.

Proof: It is similar to Theorem 7.6.

7.2.3. Algorithm for Checking Weak Cycle

With a slight difference, we have the following algorithm for checking a weak cycle. The weak cycles exist within rule sets $\mathbf{R}_{II}$ and $\mathbf{R}_{III}$.

Algorithm 7.3 Check_a_weak_cycle

Input : Type III rules, $\mathbf{R}_{III}$, Type II rules, $\mathbf{R}_{II}$, and rule $r \in \mathbf{R}_{III} \cup \mathbf{R}_{II}$

Output: Information about a weak cycle

Procedure :

```
1. $\mathbf{R}_T \leftarrow \mathbf{R}_{III} \cup \mathbf{R}_{II}$
2. P_closure(r) $\leftarrow \{r\}$
3. P $\leftarrow$ RHS(r)
4. While P_closure(r) contain unchecked rules do
5.   select a unchecked rule $r_k$ from P_closure(r) and mark it
6.   select a rule $r_i \in \mathbf{R}_T$, $r_i \neq r_k$ do
7.    if $|\text{LHS}(r_i)| \subseteq$ P
      then
8.      if $r_i \in$ P_closure(r)
        then report weak strong cycle and exit
        else
9.        P_closure(r) $\leftarrow$ P_closure(r) $\cup \{r_i\}$
10.       P $\leftarrow$ P $\cup$ RHS($r_i$)
11. stop.
```

Similarly, we have the following theorems:

Theorem 7.10 Algorithm 7.3 checks out weak cycles correctly.

Theorem 7.10 Algorithm 7.3 checks out weak cycles correctly.

Theorem 7.11 Algorithm 7.3 checks a weak cycle in time O($(K_2 + K_3) \times$ L).

Theorem 7.12 The problem of finding all weak cycles is intractable.

Since the proofs for Theorems 7.10, 7.11 and 7.12 are similar to previous ones, we omit them here.

Summary

In this chapter we have used **KB** model for knowledge bases, and **KBNet** and **RDG** as tools for describing **KB** to analyze the termination problem of a given knowledge base. Three kinds of cycles — strong cycles, semi strong cycles and weak cycles — are identified, and their characteristics are analyzed. These lead to our main result: the sufficient and necessary condition for **KB** terminating is that it does not contain any firable cycles.

These main concepts are summarized: (1) strong apple cycle; (2) semi strong apple cycle; (3) weak apple cycle; (4) strong cycle; (5) semi strong cycle; (6) weak cycle. We have also presented a set of algorithms to check cycles. The computational results are summarized in the following table:

Problem	**Computational Results**		
	strong	semi strong	weak
Find a cycle	O(K_3^3)	O($K_3 \times L$)	O($(K_2 + K_3) \times L$)
Find all cycles	Intractable	Intractable	Intractable

Table 7.1 Computational results for termination problem

These results seem rather depressing for checking the terminating condition. However, the theoretical results are not always true; there are some mitigating factors. One factor is that the cycles (strong, semi strong or weak) might be sufficiently sparse so that it is not expensive to find all such cycles. Another factor might be that in practice, once we find and eliminate a cycle in the rule set, this could possibly make checking other cycles easier. A third factor is that in a specific domain of a **KB**, there might be sufficient knowledge to rule out a large number of *cycle irrelevant* rules. More generally, when we check the existence of cycles in a *relevant rule subset*, the size of the rule set is not very large. These factors should significantly reduce the computational expense.

8.

CHAPTER 8

COMPLETENESS PROBLEM

In this chapter we discuss the completeness problem. In real life, it is rare to find an expert who possesses "complete knowledge" in a particular domain. However, our discussion of the completeness problem of a knowledge base system is based on the following fact: "a medical diagnosis expert system today does just a narrow slice of differential diagnosis" (Lenat 1990) [43]. The key factors for most of the successful knowledge base systems today are: (1) they select well-circumscribed task domains; (2) they solve narrowed or specific problems; (3) in such a domain, experts can articulate knowledge needed for high performance. We discuss the completeness problem in terms of "a narrowed slice" of a particular domain. This makes it possible to define and solve the completeness problem for a knowledge base system.

The notion of completeness of a knowledge base system has different interpretations in the literature. The classical definition for completeness is not adequate for many knowledge base applications. Previous work has oversimplified the problem. In this chapter we clarify and formalize the completeness problem of a knowledge base system in terms of the **KBS** and **KB** models. We present three perspectives of completeness problem for the knowledge base system, named as *functional completeness* (**F**_completeness), *relational completeness* (**R**_completeness) based on the **KBS** model and *rule completeness* (**r**_completeness) based on the **KB** model. We illustrate the ideas with some examples, offering complexity analysis and some theoretical results for general case.

8.1. Completeness Problem Formulation

Since there are different definitions of completeness of a knowledge base system in literature, we should clarify the completeness problem first.

8.1.1. Classical Definitions

In a formal system, such as a first order logic system, we have two classical definitions for completeness problem. The first one says that a system of inference rules is *complete* if any well formed formula (wff), called as ϕ, which logically follows from any set of axioms and/or well formed formulaes, called as Γ, are also theorems which can be derived from that set Γ. Mathematically, it can be expressed as :

$$\text{If } \Gamma \models \phi, \text{ then } \Gamma \vdash \phi.$$

Put another way, a formal system is complete if and only if any wff with logically true value in the system can be proved. Notice that this classical definition of completeness focuses on the completeness of the *reasoning mechanism.* Based on this definition, researchers have proved that modus ponens inference rule is complete, and that resolution inference rule is, or is not complete depending on the strategies applied [24, 30, 54].

The second classical definition for completeness does not depend on the inference mechanism. According to this definition, a theory for a formal system is complete if every sentence or its negation is in the theory [30]. A typical application is the Reiter's Closed World Assumption [67]. The Closed World Assumption states that if a sentence is not in a theory, then we assume that its negation is in the theory.

These classical completeness definitions are not adequate for knowledge base completeness checking for many reasons. First, many knowledge base systems are not rigorous formal systems. Second, many inference rules in knowledge base are not modus ponens or resolution principle. Third, if we take the second definition for completeness, it can "artificially complete" a system in the sense that any conclusion which is not deductible from a system could be covered by the Closed World Assumption or default rules. However, this artificial completeness will lead to the following problem: we do not know whether the lack of knowledge is caused by the domain expert, who is not able to cope with the situation, or by the knowledge engineer who did not collect the existing knowledge given by the expert. If the domain expert's knowledge in a narrowed "sliced" domain is complete, we want to exclude the omissions by the knowledge engineer's mistakes.

8.1.2. Previous Work

The problem of logical checking for the completeness of a rule-based expert system was recognized first by Suwa, Scott and Shortliffe in 1982 [85,86]. They informally and indirectly defined the problem of completeness. They described a Rule Checking Program for use with ONCOCIN, an EMYCIN-like rule-based expert system for oncology [76]. Suwa, Scott and Shortliffe do not give formal definition for completeness of a knowledge base system. Instead, they described what will happen if the knowledge base system is *not complete*:

> Incompleteness of the knowledge base is the result of missing rules. The missing rules mean that a situation exists in which a particular result is required, but no rule can succeed in that situation to produce the desired result.

Their definition differs from the classical one in the following ways: (1) in the classical definition, if a system does not miss any rule, it is still possibly incomplete; (2) based on the classical definition, the system, which misses some rules, could still be complete; e.g., a system containing the empty rule set is a complete system.

The work of Nguyen, Perkins, Laffey and Pecora [50-52] is an extension of the definition given by Suwa. Nguyen claimed that they have found four situations indicative of incompleteness, and any one of these situations might indicate that there is a rule missing. The four situations are : 1. Unreferenced attribute values; 2. Illegal attribute values; 3. Unreachable conclusions; 4. Dead-end IF conditions and dead-end goals. The definitions for incompleteness above could be stated as follows: if a knowledge base has some missing rules and/or missing facts (evidences or hypotheses) then it is incomplete. There are other similar definitions [17].

These definitions of completeness problem for a knowledge base system do not indicate what the completeness is, they tell what incompleteness is.

8.1.3. Definitions for Completeness Problem

8.1.3.1. Necessary Conditions for Completeness

To give a direct definition of a complete knowledge base system is difficult, because we first must prove that the expert has "complete" knowledge; second, we must prove that the knowledge base system "completely" encoded the expert's knowledge. The expert's knowledge should contain common sense knowledge, structural knowledge, exceptional knowledge, meta knowledge, heuristic knowledge and professional domain knowledge. The knowledge base system must encode all such knowledge and solve all possible problems in this domain at the expert level. If this is done, we have sufficient evidence to say that the **KBS** is complete. Such sufficient evidence is the sufficient condition for the **KBS** completeness. Unfortunately, so far, no such complete knowledge base system exists. The sufficient condition is too strong, and too ideal to be satisfied, at least for now. Instead of the sufficient condition for the completeness of a **KBS**, we propose to consider some necessary conditions for the completeness of a **KBS**. We will discuss the following necessary conditions: *functional complete, relational complete* and *rule complete*.

8.1.3.2. Functional Completeness

First, let us consider the **KBS** completeness problem from a functional perspective. To show that a **KBS** is complete, it is necessary to show, at least, that the **KBS** can execute all functions that are supposed to be implemented. This is functional level checking. We should identify the functions which are supposed to be implemented by this **KBS**, and test or prove this **KBS** against the specifications for those functions. The functions to be checked may be described in either the requirements or design specifications. This idea leads us to define functional completeness :

Definition 8.1: (**F**_complete) If a knowledge base system, **KBS** = (**U, T, R, F, M**), can satisfy functional specification **F**, the knowledge base system is *functional complete*, notated as **F**_complete.

One might ask, "How many knowledge base systems have functional specifications?". It is true that not all knowledge base systems have exact functional specifications during their developing phase. However, in an application field, things are much different. To design a knowledge base for a robot, we have to have exact functional specifications (actions the robotic can take), such as , "take the object," "turn it left," and so on. Considering that a knowledge base system today does only a narrow slice of a particular job, it is possible to devise functional specifications for a knowledge base system.

8.1.3.3. Relational Completeness

Next, we discuss another necessary condition for **KBS** completeness. Before knowledge can be embodied in a computer system, it must undergo a number of transformations: acquiring expertise in some domain, formalizing this expertise and encoding it. Assuming that the expert's knowledge is complete in a narrowed domain, and that this their knowledge can be described in relational table(s), if the **KBS** is complete, its rule set has to cover all expertise described in the relational table(s). Thus a very natural and necessary condition for **KBS** completeness is to determine whether the set of rules in **KBS** can cover all tuples in the relational table(s). We refer to such completeness as *relational completeness* and notated as **R**_complete.

Before we further define **R**_complete, we must make some notations and assumptions. The function *size* will be used for returning the cardinality of a set, *size*(**T**) will return the number of tuples, *size*(t_i) will return the number of elements in the tuple t_i, and *size*(**R**) will return the number of rules in the rule set **R**. We also use "| |" to represent size function, |**R**| = *size*(**R**). We assume that |(**R**)| = K and |**T**| = T.

Definition 8.2: The pair of tuples t_i and t_j in **T** are equivalent (notated as $t_i = t_j$) if and only if

(1) *size*(t_i) = *size*(t_j);

(2) t_i and t_j have the exactly same elements.

Based on the definition 8.2, we can further define the redundance of the relational table **T**.

Definition 8.3: A relational table **T** is nonredundant if and only if there are no pair of tuples t_i and t_j, such that $t_i = t_j$.

The basic assumptions for checking **R**_completeness of a knowledge base system in our framework are :

(1) A complete and nonredundant relational table T exists;

(2) A tractable process for checking rule instance exists;

Definition 8.4: A tuple $t_i \in$ **T**, $T \geq i \geq 1$, is *covered* by a rule $r_j \in$ **R**, $K \geq j \geq 1$, if and only if t_i is an instance of r_j.

Note that the term *instance* should be defined in some circumstances; the format and expression of a particular instance (tuple in relational table) vary from case to case. To simplify our following discussion, we assume that the instances (or tuples) are in an unified format. Other complicated formats of the instances will contribute to increased computational costs. For certain rule r $\in$ **R**, it is possible that the rule r can cover more than one tuples; i.e., rule r_j can cover tuples t_1, t_2, and t_3, in other words, the tuples t_1, t_2 and t_3 are instances of rule r_j. This leads us to define an instance function as follows:

Definition 8.5: For any rule r $\in$ R, its instance function, notated as **I**, will return all instance tuples of rule r.

For the previous example, **I**(r_j) = { t_1, t_2, t_3 }, here $t_1, t_2, t_3 \in$ **T**.

Definition 8.6: The rule *intersection* is defined and notated as

$$Inter(r_i, r_j) = \mathbf{I}(r_j) \cap \mathbf{I}(r_j), r_i, r_j \in \mathbf{R}.$$

For example, if **I**(r_i) = { t_1, t_2, t_3 } and **I**(r_j) = { t_2, t_3, t_5 }, then *Inter*(r_i, r_j) = { t_2, t_3 }.

Having the definitions of *cover* and *Intersection*, we can further define the *exactly cover*.

Definition 8.7: A relational table **T** is *exactly covered* by rule set **R** if and only if

(1) each tuple $t_i, t_i \in$ **T**, is covered by some rule $r_j, r_j \in$ **R**, $K \geq$ j $\geq$ 1;

(2) *Inter*(r_i, r_j) = Empty, for any pair of rules, r_i and r_j, $r_i, r_j \in$ **R**;

(3) for any rule r $\in$ **R**, if some tuple t $\in$ **I**(r), then t $\in$ **T**.

Now, we are ready to define the **R**_completeness for the knowledge base system **KBS**.

Definition 8.8: A knowledge base system **KBS** is **R**_complete with respect to relational table **T** if the rule set **R** of the **KBS** can exactly cover **T**.

8.1.3.4. Rule Completeness

Last, we should consider each rule in a knowledge base individually. If a knowledge base system is complete, each rule of its knowledge base should be *applicable*. Furthermore, if some rule of the knowledge base is not applicable, it is possibly because the rule has been implemented incompletely.

Based on the model of the knowledge base given by definition 4.2, **KB** = (**E, H, R**

), each rule r $\in$ **R** should be well defined and applicable. However, when we consider a developing process for a knowledge base, the set **E, H** and **R** evolve from empty to incomplete, then complete. For some rule in rule set **R**, during the developing phase of a **KB**, it is possible to have some evidence or hypothesis which is not defined or not satisfiable. For example, a given rule is:

$$r : e_1 \wedge e_2 ... \wedge h_1 ... \wedge h_s \rightarrow h_{i1} \wedge ... \wedge h_{it}$$

if some evidence e_i and/or hypothesis h_j belong to the left hand side of rule r, then they can not be satisfiable. This will make this rule not firable. It may indicate that:

(1) there are some missing rules in rule set **R**;

(2) there are some missing knowledge elements in sets **E** or **H**;

According to this observation, we can give the following definitions:

Definition 8.8: Given a **KB** = (**E, H, R**), r $\in$ **R**, if

(1) $e_i \in$ LHS(r), but $e_i \notin$ **E**, or

(2) $h_j \in$ LHS(r), but $h_j \notin$ **H**, or

(3) $h_j \in$ RHS(r), but $h_j \notin$ **H**

then such rule r contains some *undefined evidence* or *undefined hypothesis.*

Definition 8.9: Given a **KB** = (**E, H, R**), if for every e $\in$ **E**, $\exists$ r $\in$ **R**, such that e $\in$ LHS(r), then the **KB** is *evidence complete.*

Definition 8.10: Given a **KB** = (**E, H, R**), if for every h $\in$ **E**, $\exists$ r $\in$ **R**, such that h $\in$ LHS(r) or h $\in$ RHS(r), then the **KB** is *hypothesis complete.*

Definition 8.11: Given a **KB** = (**E, H, R**), if

(1) the **KB** is evidence complete and

(2) the **KB** is hypothesis complete and

(3) for all h $\in$ LHS(r), $\exists\, r_j \in$ **R**, such that h $\in$ r_closure(r_j)

then the **KB** is *rule complete* (r_complete).

Definitions 8.11 give us a criterion for checking a knowledge base for rule completeness.

8.2. Checking Knowledge Base Completeness

Developing a knowledge base system is an iterative process. The process of checking the completeness of a knowledge base system can be described as following loop:

```
$\mathbf{KBS}_m$ := KBS
While not complete( KBS ) do
  $\mathbf{KBS}_m$ := Modify( KBS )
Acceptable_KBS := $\mathbf{KBS}_m$.
```

The predicate *complete* could be **F**_complete, **R**_complete, **r**_complete or a combination of the three, according to the request. Checking the completeness of a knowledge base system is an iterative process.

In this section we discuss two examples for the knowledge base completeness checking. The first example shows that if the application domain has the *recursive enumerable* characteristic (to be defined), we can use a simple formal approach to check its **F**_completeness. The second one is that if the relational table is available, based on the table, we can check the **R**_completeness of the corresponding knowledge base system. The two examples are used to further explain our ideas and to derive some general results.

8.2.1. Checking Functional Completeness

There are two fundamental methods in checking **F**_completeness of a knowledge base system, namely, formal proving and testing. In the method of formal proving, as we will describe later, we prove the effectiveness desired. In the method of testing, we should select functionally important test cases which can be used for testing each function of a

KB. We select a simple example for explaining the concept of F_completeness. In some application domain, the functional specifications can be clearly partitioned, the inputs can be encoded as strings of characters, or bags of characters, and there exists a procedure (note that the procedure may be non halting) that lists all possible input strings of U in some order. We call this kind of domain as *recursive enumerable* [36].

If the application domain of a knowledge base system is recursive enumerable, we use a somewhat formal method to prove its F_completeness. To explain how these work, let us use the simple knowledge base system **BAGGER**, which is defined in Chapter 4 by Winston [91].

8.2.1.1. Domain for BAGGER System

For the **BAGGER** system, we can use the notations :

B : represents a big item

M : represents a medium item

S : represents a small item

For any application instance of this domain, it can be represented as

$B^n M^m S^k$ (Here, m, n, k ≥ 0),

This means that there are n large items, m medium items and k small items to be bagged. The "recursive enumerable" is obvious since all instances of the application domain can be simply represented by one formula.

8.2.1.2. Functional Specification and Rule Set for BAGGER System

The functional specifications for BAGGER system, **F**, is:

F : To bag all available items.

The input space can be partitioned and represented as:

IN_1 : Available items from one type (large, medium or small);

IN_2 : Available items from two types;

IN_3 : Available items from three types.

The rule set for **BAGGER** is **R** = { $r_1, r_2, ..., r_7$ }. The simplified representation for the set of rules is:

$$r_1 : e_1 \wedge e_2 \rightarrow h_1$$
$$r_2 : e_3 \wedge e_4 \rightarrow h_2$$
$$r_3 : e_5 \wedge e_6 \rightarrow h_3$$
$$r_4 : e_1 \wedge e_7 \rightarrow h_4$$
$$r_5 : e_2 \wedge e_8 \rightarrow h_4$$
$$r_6 : e_3 \wedge e_9 \rightarrow h_4$$
$$r_7 : e_{10} \rightarrow h_5$$

8.2.1.3. Checking F_completeness

According to the characteristics of the application domain, we can easily conclude the following:

Fact : The knowledge base system BAGGER is functional complete.

Proof : According to the functional specification, F =: to bag all available items and the decomposed input space IN_1, IN_2 and IN_3, the application instances can be in any one of the cases:

Case 1 : There are only some large (or medium, or small) items to be bagged.

Case 2 : There are only two types of items to be bagged.

Case 3 : There is a combination of the three types of items to be bagged.

For Case 1, if the items are all large, the **BAGGER** just uses rules r_1 and r_4 repeatedly; with finite number of times, the packing job can be finished. Similarly, the **BAGGER** just uses r_2 and r_5 for packing all medium items and r_3 and r_6 for packing all small items.

For Case 2, if the application instance contains only large and medium items, the **BAGGER** can first use rules r_1 and r_4 until all the large items are bagged, then use rules

r_2 and r_5 a finite number of times, until the bagging job is completed. Similar situation would exist if the application instance contains only large and small items, or medium and small items.

For Case 3, the BAGGER should use rules r_1 and r_4 to bag all large items, then use r_2 and r_4 to bag all medium items, and finally use r_3 and r_6 to finish the bagging job. The bagging job can be completed in finite steps. □

8.2.1.4. Comments

There is a very natural relationship between software engineering and artificial intelligence, particularly when we consider the relation of the **F_complete** of a knowledge base system to functional testing in software engineering. Many techniques used in software engineering [37, 57] can be borrowed for checking the **F_completeness** of the knowledge base system. As we mentioned before, two basic approaches can be used for checking functional completeness, formal proving and data testing. In the example above, we wanted to show how the idea for checking **F_complete** works; we do not attempt an exhaustively explanation of how to implement the checking of **F_complete**. Slagle [80] proposes a methodology for creating a *knowledge specification* using conceptual structures [81]. The preliminary experimental work is reported. Both functional testing and empirical data testing are big topics. If a knowledge base system is a research project, it is very hard to devise the functional specification (or knowledge specification) for it, either at the beginning or ending of the project. The lack of the testable functional specification is a serious problem [19, 20, 65]. We do not want to discuss these issues in any detail here.

8.2.2. Checking Relational Completeness

According to the definition of **R_completeness**, proving whether a given **KBS** is **R_complete** requires checking to see if the rule set **R** of the **KBS** exactly covers the relational table **T** of the KBS. In this section, we will first explain how this idea works through the 8-puzzle problem, then we will discuss the general situation, and last we will consider the optimality problem for **R**_completeness in the sense that the set of rules of a **KBS** has minimal size.

8.2.2.1. 8_Puzzle Problem

Suppose we want a knowledge base system to solve the 8-puzzle problem. We use the characters **A, B, ..., I** to represent the nine positions on the board as shown in Figure 8.1.

A	B	C
D	E	F
G	H	I

Figure 8.1 8-puzzle problem.

For some input configuration of 8 tiles, the goal configuration is presented in Figure 8.2.

1	2	3
8		4
7	6	5

Figure 8.2 The goal for 8-puzzle problem.

8.2.2.2. A KBS for 8_puzzle Problem

A knowledge base system **KBS** = (**U, T, R, F, M**), for the 8-puzzle problem, may be described as follows:

U : tiles : 1, 2, 3, 4, 5, 6, 7, 8

positions : **A, B, C, D, E, F, G, H, I**

move actions : left, right, up, down.

T : the relational table shows how the tiles move with respect to the empty position on the puzzle board.

R : the given set of rules which are used to find possible movements of certain tiles. We will give them later.

F : for some input configurations, the **KBS** should give the sequence to achieve the goal configuration in Figure 8.2; for some other starting states, the **KBS** should prompt memory space limitation and show its effort.

M : the mechanism for selecting movements, which can be A or A^* algorithms as Nilsson proposed and the input/output procedures [54].

Tuple_no	Empty_Position	Action
1	A	Tile in B moves left
2	A	Tile in D moves up
3	B	Tile in A moves right
4	B	Tile in C moves left
5	B	Tile in E moves up
6	C	Tile in B moves right
7	C	Tile in F moves up
8	D	Tile in A moves down
9	D	Tile in E moves left
10	D	Tile in G moves up
11	E	Tile in B moves down
12	E	Tile in D moves right
13	E	Tile in H moves up
14	E	Tile in F moves left
15	F	Tile in C moves down
16	F	Tile in E moves left
17	F	Tile in I moves up
18	G	Tile in D moves down
19	G	Tile in H moves left
20	H	Tile in G moves right
21	H	Tile in I moves left
22	H	Tile in E moves down
23	I	Tile in H moves right
24	I	Tile in F moves down

Table 8.1 Relational table for 8-puzzle problem.

Now we face three problems:

(1) How to create a set of rules **R** based on relational table **T**;

(2) For the given relational table **T** and a set of rules **R**, how to check whether the **R** exactly covers **T**;

(3) Assuming that there is more than one set of rules, $\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_h$, $h > 1$, such that all of them exactly cover the given relational table **T**, which of $\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_h$ is the "optimal" set of rules?

In this dissertation, we do not consider the first problem. The task to create a set of rules **R** for a **KBS** belongs to system builder; we assume that a relational table **T** and a set of rules **R** are given. We concentrate on the second and third problems.

8.2.2.3. Checking R_completeness

Our discussion starts from the 8-puzzle problem. If a given set of rules **R** has 24 rules, each of which exactly corresponds to each tuple from relational table **T** in Figure 8.3, the task to check whether **R** exactly covers **T** becomes very easy. If the relational table **T** is complete and nonredundant, each tuple of **T** can be viewed as a rule, which is a much simplified situation. Actually, this is the basic ideas presented by Suwa and Cragun [17,85].

Let us consider a general situation. Assume that the cardinality of relational table **T** is T, the cardinality of rule set **R** is K, and K is much less T. For instance, in 8-puzzle problem, $T = 24$. Consider that the rule set contains 3 rules, one of which to deal center position being empty and one of which to deal with corner position empty. The third rule covers the rest of the cases. In order to check whether **R** exactly covers **T**, we give the following algorithm:

Algorithm 8.1 Checking R_completeness

Input: A relational table **T**, |**T**| = T, a set of rules **R**, |**R**| = K.

Output: If **R** exactly covers **T**, return "Yes," otherwise "No."

Procedure :

```
0: $R_1 \leftarrow \mathbf{R}$;
1: While $R_1 \neq$ Empty do
     begin
      select one rule r from $R_1$;
      find and mark all instance tuples for r from T;
      put all instances tuples for r into I( r );
      $R_1 \leftarrow R_1$ - { r }
     end;
2: If $\exists$ unmarked tuples in T then return "No"
        else
3:         If $\Sigma$ |I(r)| $\neq$ |T| for all rules, r $\in$ R
           then return "No"
           else return "Yes;"
4: end.
```

8.2.2.4. Complexity Analysis

In order to analyze the complexity of the algorithm for checking **R**_complete, we need more information about the rule, the tuple and the items in the rule. Assume the average size for a tuple in **T** is L_t, the average size for a rule in **R** is L_r, and the average number of elements, which can be covered by a condition or an action in rule r, is L_e. Now, we can analyze the complexity for the Algorithm 8.1.

Step 1 : For finding all instance tuples for a rule from relational table **T**, we need $T \times L_t \times L_r \times L_e$ comparisons in worst case. Thus, for finding all instance tuples for the rule set **R**, we need $O(K \times T \times L_t \times L_r \times L_e)$ comparisons.

Step 2: To determine whether the rule set **R** covers relational table **T**, we need at most $O(T)$ operations for checking if there are some unmarked tuples.

Step 3: For checking the intersection of any pair of rules in **R**, we need $O(T)$.

So, the worst case complexity for Algorithm 8.1 is

$$O(T + T + K \times T \times L_t \times L_r \times L_e).$$

If (1) L_t, L_r, and L_e can be represented as L; (2) $K = C \times T$, $1 \le C > 0$, the complexity becomes $O(T^2 \times L^3)$. We have the following theorem:

Theorem 8.1: The algorithm 8.1 checks R_completeness in time $O(T^2 \times L^3)$.

8.2.3. NP Result for Optimal Set of Rules

We define an "optimal" set of rules as those that are most general and concise; in other words, an optimal set of rules contains a minimal number of rules. This problem, which we call the *minimal exact rule cover problem* (**MERC**), can be formulated as follows:

Instance: Given a finite set of elements, $\mathbf{U} = \{ u_1, u_2, ..., u_n \}$, a relational table **T** based on **U**, $\mathbf{T} = \{ t_1, t_2, ..., t_T \}$, a set of rules $\mathbf{R} = \{ r_1, r_2, ..., r_K \}$, and each rule $r \in \mathbf{R}$ can cover one or more tuples in **T**, positive integer $N \le |\mathbf{R}|$.

Question: Does **R** contains a cover for **T** of size N or less, i.e., a subset $\mathbf{R}_1 \subseteq \mathbf{R}$ with $|\mathbf{R}_1| \le N$ such that every tuple of **T** is covered by at least one rule of $\mathbf{R}_1$?

By comparing the *minimal exact rule cover problem* (**MERC**) with **MINIMUM COVER** problem [27], which is NP_complete, we can find that there is a one-to-one corresponding transformation.

MINIMUM COVER PROBLEM [27]: (pp 222, Computers and Intractability)

Instance: Collection C of subsets of a finite set S, positive integer $K \le |C|$.

Question: Does C contain a cover for S of size K or less, i.e., a subset $C' \subseteq C$ with $|C'| \le K$ such that every element of S belongs to at least one member of C' ?

So, we can have the following result:

Theorem 8.2: The Minimal Exact Rule Cover Problem is NP_complete.

Proof : To show the **MERC** problem is NP_complete, we can look at the relational table **T** as a finite set of tuples, and **R** as a collection of subsets of **T**. It is obvious that there are one-to-one corresponding transformation between **MERC** problem and **MINIMUM COVER Problem** [27]. The **MINIMUM COVER Problem** is NP_complete, thus the theorem is proved. □

Actually, the theoretical result that **MERC** problem is NP_complete is devastating. This like the NP result for traveling salesman problem has never prevented salesmen from making decisions. The significant benefit of this NP result is to persuade us to avoid pursuing exact optimality and to try to obtain reasonable solutions. In the context of knowledge base systems, from a practical point of view, the size of the rule set is not always a serious problem, except when memory and disk space are limited.

8.2.4. Checking Rule Completeness

The purpose in checking for rule completeness of a **KB** is to determine whether all evidences and hypotheses defined are useful, and whether any rules of the **KB** contains undefined elements in its left hand side and right hand side. The following Algorithm 8.2 is used for checking rule completeness; it is based on definition 8.11.

Algorithm 8.2 Check Rule Completeness

Input : **E, H, R**

Output: Information about rule completeness

Procedure:

```
1. n ← |E|, m ← |H|, k ← |R|
2. Arrange the evidences in E as $e_1, ..., e_n$.
   Arrange the hypotheses in H as $h_1, ..., h_m$
   Arrange the rules in R as $r_1, ..., r_k$.
```

```
3. For i = 1 to n do
     e ← $e_i$
     for j = 1 to k do
       if e ∈ LHS($r_j$)
        then Evidence_complete($e_i$) ← true;
        else if j = k
            then Evidence_complete($e_i$) ← false;
4. For i = 1 to m do
     h ← $h_i$
     for j = 1 to k do
       if h ∈ LHS($r_j$) ∪ RHS($r_j$)
        then Hypotheses_complete($e_i$) ← true;
        else if j = k
            then Hypotheses_complete($e_i$) ← false;
5. For i=1 to k do
     If ( LHS($r_i$) ⊂ E and RHS($r_i$) ⊂ H ) or
          ( LHS($r_i$) ⊂ (E ∪ H) and RHS($r_i$) ⊂ H )
      then Complete($r_i$) ← true
      else Complete($r_i$) ← false;
6. Stop.
```

The correctness of this algorithm is obvious. The complexity of algorithm 8.2 is analyzed as follows: The step 3 costs O($N \times K$), step 4 costs O($M \times K$) and step 5 costs O($K \times L$). The L is the average cardinality of the rule in rule set **R**. So the complexity of algorithm 9.2 is :

$$O((N \times K) + (M \times K) + (K \times L)) = O(K \times (N + M + L))$$

We have :

Theorem 8.3: The algorithm 8.2 checks rule completeness in time $O(K \times (N + M + L))$.

Summary

It has been said that a key feature lacking in knowledge base systems is "a suitable verification methodology or a technique for testing the consistency and completeness of a rule set" [35]. In this chapter we discussed the completeness problem for a knowledge base system. Based on model **KBS**, we defined the completeness problem for a knowledge base system, namely *functional completeness* (**F**_complete) and *relational completeness* (**R**_complete); and we defined *rule completeness* (**r**_complete), based on a **KB** model. Under this formalization, we demonstrated how to check them for a knowledge base system and obtained some theoretical results.

9.

CHAPTER 9

CONSISTENCY PROBLEM

This chapter focuses on the *consistency* problem of knowledge base systems, which is an important property in terms of our effectiveness definition. The "classical" definition for consistency of a formal system is inadequate for many cases in a knowledge base system context. Earlier works have only roughly defined the consistency problem of a knowledge base system, and given some simplified results. We will use the qualitative model **KB** and some rule properties of the rule set of the **KB** to check the consistency problem. Based on this model, we will identify and clarify the consistency problem of a knowledge base and present a model-based methodology for checking and analyzing the consistency of a given knowledge base.

9.1. Considerations

One difference between knowledge base systems and conventional programs is the acquisition of knowledge during program lifetime. Usually the knowledge base systems deal with incomplete knowledge and that is why additional knowledge, entered by the user, has to fit into the already existing base. The "fit" means that the knowledge base has to remain consistent. Consistency checking is routine work and involves deleting, modifying and adding knowledge. Thus checking and analyzing the consistency of the knowledge base is a very important issue for knowledge base application. In the "classical" definition for consistency in a formal system, a formal system is consistent if and only if no contradiction exists; more concretely, a formal system is consistent if it is impossible to prove both an assertion P and its negation ¬P [24, 30, 54]. This definition is inadequate for many cases within the knowledge base system context.

It is important to point out the differences in the notion of consistency as employed in a formal logic and as used in the rule-based knowledge systems context. In terms of formal logic, the set of propositions

$$e_1 \rightarrow h_1$$
$$e_1 \wedge e_2 \rightarrow \neg h_1$$

is consistent if e_1 is assigned false. This means that both propositions are true and no contradiction is entailed. If a knowledge base contains the preceding propositions as rules, it is possibly inconsistent: if e_1 and e_2 were to be given true evidences, the knowledge base would be ready to assert both h_1 and $\neg h_1$. If e_1 or e_2 is always false for all problems, such a rule(s) is actually useless. Furthermore, if the evidences e_1 and e_2 can not be given at a particular time, or if they are mutually exclusive, the two propositions (rules) do not lead to an inconsistency because the rule $e_1 \wedge e_2 \rightarrow \neg h_1$ can never be fired. In other words, when we consider the consistency problem of a knowledge base, we should take into account the particular situations. Thus, consistency in a knowledge base is more complicated than it is in formal logic. We would check the consistency of a knowledge base from evidence, hypothesis and rule perspectives.

In the next section, based on the model of knowledge base, **KB**, we define *evidence consistency, hypothesis consistency and rule consistency.* Then we develop algorithms for checking the consistency of a given knowledge base and present complexity analysis.

We assume the existence of the following conditions:

- The **KB** is non-redundant;
- The **KB** is acyclic;

We identified the varieties of redundant rules: *self redundant rule, redundant rule, subsumed redundant rules* and *redundant rule triangle* when we discuss the minimality problem of a knowledge base system in Chapter 6. Also, we have identified the *strong cycle, semi strong cycle* and *weak cycle* when we discussed the terminating problem of a knowledge base in Chapter 7. So, when we consider the consistency problem, we have already checked and eliminated the redundant rules, and checked and solved the cycling problem. Therefore, we can reasonably assume that the **KB** is both non-redundant and acyclic. These assumptions make our work much different from the previous work, which found the redundant rules to be a main source of inconsistency of the **KB** [17,50-52,85,86].

9.2. Consistency Problem of Knowledge Base

In this section, we identify and clarify the consistency problem of knowledge base. Based on the **KB** model, we define evidence consistency, hypothesis consistency and rule consistency.

9.2.1. Evidence Consistency

Given a knowledge base **KB** = (**E, H, R**), some possible inputs for knowledge base **KB** are notated as E_1, E_2, ..., E_p, and $E_i \subset$ **E**, for $p \geq i \geq 1$. These inputs should not violate any of the semantic constraints that explicitly or implicitly exist for the application domain. There are general constraints, type constraints, logical constraints and domain dependent constraints.

General constraints : For example, the propositions e_i and e_j are :

e_i : Smith is a male human being.

e_j : Smith is pregnant.

If both e_i and e_j belong to a rule's left hand side, $e_i \in$ LHS(r) and $e_j \in$ LHS(r), r $\in$ **R**, this rule r violates the general constraints : a male cannot be pregnant.

Type constraints : As an example, a person has one and only one age (at a given time), or if the evidence is represented by integers but the given value is a real, this violates the type constraint.

Logical constraints : An individual cannot logically be assigned the properties of both male and female, nor can an object belong to both animal and vegetable categories.

Domain dependent constraints : As an example, we have a knowledge base system which is used in a medical application. The propositions e_i and e_j are used to represent symptom_1 and symptom_2 respectively, if it is impossible for both symptom_1 and symptom_2 to exist at the same time in the medical domain, but such evidences are encoded in a rule r, { e_i, e_j } $\subset$ LHS(r), r $\in$ **R**. This rule violates the domain constraints.

Considering the evidence consistency or semantic constraints above, the **KB** relates to additional evidence constraints $\mathbf{C_E}$. The $\mathbf{C_E}$ is a set of pairwise evidences of **E**, which are incompatible. We can derive the following definition:

Definition 9.1: A knowledge base, **KB** = (**E, H, R**), is *evidence consistent* if any pair of evidences in any rule, r $\in$ **R**, does not appear in the semantic constraints $\mathbf{C_E}$.

9.2.2. Hypothesis Consistency

Now let us consider the hypothesis consistency of a given **KB** = (**E, H, R**). As we defined, the hypotheses can appear within all three type's rules, either on the left hand side or right hand side of a rule. The hypotheses may be actions, decisions, diseases or some intermediate solutions, depending on the particular application domain.

Definition 9.2: Given a **KB** = (**E, H, R**), and two hypotheses $h_i, h_j \in$ **H**, if $h_i = \neg h_j$, we say that the pair of hypotheses, h_i and h_j, are *contradictory*.

The meaning of contradiction is obvious. Sometimes, a pair of hypotheses, h_i, and h_j are not contradictory, i.e., $h_i \neq \neg h_j$, for instance, let h_i and h_j represent disease_1 (d_1) and disease_2 (d_2) respectively. But from the doctor's (domain expert) point of view, disease_1 (d_1) and disease_2 (d_2) are *incompatible*, which means that a patient can suffer disease_1 (d_1) or disease_2 (d_2), but not both of them at same time. This situation leads to the following definition :

Definition 9.3: Given a **KB** = (**E, H, R**), and two hypotheses h_i, $h_j \in$ **H**, if h_i and h_j can not be true in the same time, we say that the pair of hypotheses are *incompatible*.

Based on the concepts of contradictory and incompatible, we define the term *conflicting* :

Definition 9.4: Given a **KB** = (**E, H, R**), and two hypotheses h_i, $h_j \in$ **H**, if h_i and h_j are either contradictory or incompatible, then we say that the two hypotheses, h_i and h_j are *conflicting*.

Similarly, with evidence constraint C_E, the knowledge base **KB** is also related to hypothesis constraint C_H. The C_H contains pairs of hypotheses of **H** which are conflicting. So we have:

Definition 9.5: A **KB** is *hypothesis consistent* if any pair of hypotheses in any rule, r $\in$ **R**, does not appear in hypothesis constraint C_H.

9.2.3. Rule Consistency

We are ready to discuss the rule consistency problem in terms of the concepts of hypothesis consistency and rule closure as defined in Chapter 5.

9.2.3.1. Directly Conflicting Rules

Consider a pair of rules, r_i, $r_j \in$ **R**, if the given evidences, which are evidence consistent, can satisfy both left hand sides of the two rules, and the firing of the two rules will immediately lead to conflicting conclusions, the pair of rules are directly conflicting. The formal definition is :

Definition 9.6 A pair of rules $r_i, r_j \in$ R, are *directly conflicting* if they have the following properties :

(1) LHS(r_i) $\subseteq$ LHS(r_j) and RHS(r_i) = $\neg$ RHS(r_j), or

(2) LHS(r_i) $\subseteq$ LHS(r_j), RHS(r_i) and RHS(r_j) are incompatible.

9.2.3.2. Potentially Conflicting Rules

If a pair of rules, $r_i, r_j \in$ **R**, do not immediately lead to a conflicting conclusion, it is possible to lead indirectly to a conflicting conclusion.

Definition 9.7 A pair of rules $r_i, r_j \in$ R, are *potentially conflicting* if they have the following properties :

(1) LHS(r_i) $\subseteq$ LHS(r_j), LHS(r_i) $\overset{*}{\Rightarrow}$ X, LHS(r_j) $\overset{*}{\Rightarrow}$ $\neg$ X or

(2) LHS(r_i) $\subseteq$ LHS(r_j), LHS(r_i) $\overset{*}{\Rightarrow}$ X, LHS(r_j) $\overset{*}{\Rightarrow}$ Y, X and Y are incompatible.

Example 9.1 The set of rules in R are :

$r_1 : e_1 \rightarrow h_1$

$r_2 : e_1 \wedge e_2 \rightarrow \neg h_1$

$r_3 : e_3 \rightarrow h_2$

$r_4 : h_2 \rightarrow h_3$

$r_5 : e_3 \wedge e_4 \rightarrow h_5$

$r_6 : h_5 \rightarrow \neg h_3$

In this example, r_1 and r_2 are directly conflicting rules, and r_3 and r_5 are potentially conflicting rules.

Definition 9.8 A **KB** is *rule consistent* if it does not contain any directly conflicting rules and potentially conflicting rules.

The definitions 9.1, 9.4 and 9.8 form our basis for checking knowledge base consistency. We give the following definition:

Definition 9.9 A **KB** is *consistent* if it is

(1) evidence consistent and

(2) hypothesis consistent and

(3) rule consistent.

9.3. Algorithms for Checking Consistency of a KB

In this section we develop our algorithms for checking the consistency defined in the previous section. To check evidence consistency and hypothesis consistency, we need some knowledge about the application domain. To check the rule consistency of the **KB**, we can check the rule set in terms of its totally dependent property. We assume that the domain experts can provide evidence constraint $\mathbf{C_E}$ and hypothesis constraint $\mathbf{C_H}$.

Like the process for checking the completeness of a knowledge base system, the process for checking the consistency of a **KB** is an iterative loop:

```
$\mathbf{KB_m} = \mathbf{KB}$
While not consistent( $\mathbf{KB_m}$ ) do
    $\mathbf{KB_m}$ := Modify( $\mathbf{KB_m}$ )
Acceptable_KB := $\mathbf{KB_m}$.
```

The predicate *consistent* could be *evidence consistent*, *hypothesis consistent*, *rule consistent* or a combination, according to the particular need.

9.3.1. Algorithm for Checking Evidence Consistency

From a theoretical point of view, to determine the evidence consistency of a knowledge base system, we have to check the validity of each possible subset of evidences. If $|\mathbf{E}| = n$, the total number of the possible subsets of input evidences is 2^n, and the complexity of checking evidence consistency is exponential. We propose to check the evidence consistency in another way: to check the rules, which belong to Type I and Type II, one by one, based on the evidence constraint $\mathbf{C}_\mathrm{E}$.

Algorithm 9.1 Check evidence consistency

Input : $\mathbf{R}_\mathrm{I}$ (or $\mathbf{R}_\mathrm{II}$)

Output: Information about evidence consistency

Procedure:

```
1. Temp ← $\mathbf{R}_\mathrm{I}$
2. while Temp $\neq \emptyset$ do
   2.1 pick a rule r from Temp
   2.2 Temp ← Temp - { r }
   2.3 while LHS(r) $\neq \emptyset$ do
       pick a evidence $e \in$ LHS(r)
       LHS(r) ← LHS(r) - $\{e\}$
       comparing $e$ with each $e_x \in$ LHS(r)
       if $\{e, e_x\} \in C_E$
          then evidence_inconsistent($e$, $e_x$) ← true
3. end.
```

This algorithm can be used for checking evidence consistency of Type I and Type II rules. The complexity of checking Type I rules is $O(|\mathbf{R}_\mathrm{I}| \times L_e^2)$, or $O(K_1 \times L_e^2)$, the L is the average cardinality of LHS(r), r $\in \mathbf{R}_\mathrm{I}$; for checking Type II rules, the complexity is

O(|$\mathbf{R}_{II}$| × L_e^2), or O($K_2 \times L_e^2$), the L_e is the average cardinality of LHS(r).evid, r ∈ $\mathbf{R}_{II}$. We can use hash table to store the information of $\mathbf{C}_E$, this leads that the checking whether {e, e_x } ∈ $\mathbf{C}_E$ only costs O(1) step. So, we have the following result:

Theorem 9.1: The algorithm 9.1 checks evidence consistency of a **KB** in time O(K_i × L_e^2), i = 1 or i = 2.

9.3.2. Algorithm for Checking Hypothesis Consistency

For checking hypothesis consistency, we give following algorithm:

```
Algorithm 9.2 Check hypothesis consistency
Input : $\mathbf{R}_I$ ( $\mathbf{R}_{II}$ or $\mathbf{R}_{III}$ )
Output: Information about hypothesis consistency
Procedure:

  1. Temp ← $\mathbf{R}_I$
  2. while Temp ≠ ∅ do
     2.1 pick a rule r from Temp
     2.2 Temp ← Temp - { r }
     2.3 while RHS(r) ≠ ∅ do
         pick a hypothesis h ∈ RHS(r)
         RHS(r) ← RHS(r) - {h}
         comparing h with each $h_x$ ∈ RHS(r)
         if {h, $h_x$ } ∈ $C_H$
            then confliction_hypo(h, $h_x$) ← true
  3. end.
```

This algorithm can be used for checking the rules of all types. For checking Type I rules, the complexity is O(|$\mathbf{R}_I$| × L_h^2), or O(K_1 × L_h^2) here the L is average cardinality of RHS(r), r ∈ $\mathbf{R}_I$. For checking Type II rules and Type III rules, the complexity is O(|$\mathbf{R}_{II}$| × L_h^2) and O(|$\mathbf{R}_{III}$| × L_h^2) respectively, the L is average size of RHS(r)+LHS(r).hypo, r ∈

$\mathbf{R}_{\mathbf{II}}$ and/or r $\in$ $\mathbf{R}_{\mathbf{III}}$.

Theorem 9.2: Algorithm 9.2 checks hypothesis consistency of a **KB** in time O($K_i \times L_h^2$), i = 1, 2, or 3.

9.3.3. Algorithms for Checking Directly Conflicting Rules

According to the size of "IF part" of rules in a given **KB**, we can divide the set of rules into q groups, G_1, G_2, ..., G_q. The *i* of G_i indicates that the size of "IF part" of all rules in group G_i is i. From the definition of directly conflicting rules, we make the following observation: If r_x, $r_y \in$ **R** and they are directly conflicting rules, and $r_x \in G_l$, $r_y \in G_m$ then $l \leq m$. Algorithm 9.3 is based on this observation.

Algorithm 9.3 Check directly conflicting rules

Input : $\mathbf{R}_{\mathbf{I}}$, groups of production rules, G_1, G_2, ..., G_q.

Output: Information about directly conflicting rules

Procedure:

```
1.  Let S = { $G_1, G_2, ..., G_q$ }, p ← |S|
2.  for i =  1  to  p - 1  do
    2.1  for each rule $r_x$ from $G_i$
    2.2    for j = i+1 to p do
    2.3      for each rule $r_y$ from $G_i$ do
    2.4        if LHS( $r_x$ ) $\subseteq$ LHS( $r_y$ )
               then if RHS( $r_x$ ) = $\neg$ RHS( $r_y$ ) or
                    RHS( $r_x$ ) and RHS( $r_y$ ) are incompatible
                      then Direct_Confliction($r_x$, $r_y$) ← true
3.  end.
```

Notice that we arrange the rules in an arbitrary order r_1, r_2, ..., r_s, in group G_t, $1 \leq t \leq q$. The rule $r_x \in G_i$, $r_y \in G_{i+1}$, $1 \leq i \leq p-1$, the subscriptions of rules, "x" and "y" are in the ranges $1 \le x \le |G_i|$ and $1 \le y \le |G_{i+1}|$ respectively. If we assume that the total number of rules in rule set **R** is *K*, $|\mathbf{R}| = K$, and the average number of rules in the group G_i, $1 \le i \le q$, is h, $K/q = h$, then the complexity of the algorithm 9.3 in the worst case is :

$$\begin{aligned} & h^2 \times ((q-1) + (q-2) +, \ldots, + 1) \\ &= (h^2 \times (q-1) \times q)/2 \\ &= ½ (K^2/q^2) \times q \times (q-1) \\ &\approx ½ K^2 \\ &= O(½K^2). \end{aligned}$$

So we have the following theorem :

Theorem 9.3: The algorithm 9.3 checks directly conflicting rules in time $O(½K^2)$.

9.3.4. Algorithm for Checking Potentially Conflicting Rules

Since the **KB** is acyclic and nonredundant, we can first use the following algorithm to find the r_closure(r) for each rule $r \in$ **R**. Then we can use the r_closure algorithm for checking potentially conflicting rules.

```
Algorithm 9.4 r_closure:
Input : $\mathbf{R_I}$, $\mathbf{R_{III}}$
Output: r_closure(r) for each rule $r \in \mathbf{R_I}$ or $\mathbf{R_{III}}$.
Procedure:

  1. Temp1 $\leftarrow \mathbf{R_I}$
  2. Temp2 $\leftarrow \mathbf{R_{III}}$
  3. while Temp1 $\neq \emptyset$ do
     3.1 pick a rule r from Temp1
     3.2 Temp1 $\leftarrow$ Temp1 - { r }
     3.3 r_closure(r) $\leftarrow$ RHS(r)
     3.4 while Temp2 $\neq \emptyset$ do
```

```
        pick a rule $r_x$ from Temp2
        Temp2 $\leftarrow$ Temp2 - { $r_x$ }
        if LHS($r_x$) $\subseteq$ r_closure(r)
          then r_closure(r) $\leftarrow$ r_closure(r) $\cup$ RHS($r_x$)
4. Stop
```

The complexity of this r_closure algorithm is O($|\mathbf{R}_\mathrm{I}| \times |\mathbf{R}_\mathrm{III}|$), or O($K_1 \times K_3$). The following algorithm 9.5 directly uses the algorithm 9.4 to check potentially conflicting rules.

Theorem 9.4 : The algorithm 9.4 calculates the r_closure for all rule r $\in$ $\mathbf{R}_\mathrm{I}$ in time O($K_1 \times K_3$).

By using algorithm 9.4 for calculating the rule r_closure, we give the following algorithm to check potentially conflicting rules.

```
Algorithm 9.5 Check potentially conflicting rules
Input : The r_closure(r) for each rule r $\in$ $\mathbf{R}_\mathrm{I}$
Output: Information of potentially conflicting rules
Procedure :  { the algorithm r_closure(r) given }

 1. For each pair of rules, $r_i, r_j \in R_I$,
      If LHS($r_i$) $\subseteq$ LHS($r_j$)
        then
        X $\leftarrow$ r_closure($r_i$);
        Y $\leftarrow$ r_closure($r_j$);
 2.     If (X = $\neg$ Y) or (X and Y are incompatible)
         then Potential_Confliction($r_i, r_j$) $\leftarrow$ true
 3. Stop.
```

The complexity of this algorithm in the worst case is O($|\mathbf{R}_\mathrm{I}|^2$), or O(K_1^2). Together with the algorithm 9.4, we have the theorem:

Theorem 9.5: The algorithm 9.5 checks potentially conflicting rules in time O($K_1^2 + K_1 \times K_3$).

Summary

In this chapter, we have focused on the consistency problem of a given knowledge base **KB** and identified three kinds of consistency problems : *evidence consistency, hypothesis consistency* and *rule consistency*. A consistent knowledge base should satisfy these consistencies. We also designed a set of algorithms for checking such consistencies of the knowledge base and gave some results of their complexities.

10.

CHAPTER 10

CONCLUSION

We summarize the main contributions of this dissertation to the methodology and theory of checking the effectiveness of knowledge base systems and knowledge bases:

(1) We presented qualitative models for knowledge base systems and knowledge bases, **KBS** and **KB**, and the related tools, **KBNet** and **RDG**. These domain independent models with the tools can be used for describing and analyzing the structures of knowledge bases. The two models provide a framework that helps in analyzing rule dependency and rule properties and in formalizing effectiveness problems.

(2) We provided a set of definitions for minimality, termination, completeness and consistency of knowledge bases in the context of our models for knowledge base systems and knowledge bases. These definitions provide a systematic treatment for the effectiveness of knowledge base systems.

(3) We designed a set of algorithms for checking minimality, termination, completeness and consistency problems, and analyzed their computational complexities.

The Table 10.1 compares our major contributions with previous works.

Model	Problem	Suwa	Nguyen	Cragun	Ginsberg	Zhao
KBS	Functional Completeness	No	No	No	No	Yes
	Relational Completeness	Yes	Yes	Yes	No	Yes
	Minimal Exact Cover Rule Set	No	No	No	No	Yes
KB	Minimality					
	Self Redundant Rules	No	No	No	No	Yes
	Redundant Rules	Yes	Yes	Yes	Yes	Yes
	Subsumed Rules	Yes	Yes	Yes	Yes	Yes
	Redundant Triangle	No	No	No	No	Yes
KB	Termination					
	Strong Cycle Rules	No	No	No	No	Yes
	Semi Strong Cycle Rules	No	No	No	No	Yes
	Weak Cycle Rules	No	No	No	No	Yes
KB	Rule Completeness	No	Yes	Yes	Yes	Yes
KB	Consistency					
	Evidence Consistency	No	No	No	No	Yes
	Hypothesis Consistency	No	No	No	No	Yes
	Rule Consistency Directly Conflicting	Yes	Yes	No	Yes	Yes
	Rule Consistency Potentially Conflicting	No	No	No	Yes	Yes

Table 10.1 Comparisons with other works

As we mentioned before, the work in this field is very difficult, and so far very few works have been reported. A major obstacle in this field seems to be the difficulty in coming up with precise definitions for problems and properties. Our model-based methodology can be considered as a necessary ingredient in a general theory for checking the effectiveness of knowledge base systems and knowledge bases.

There is considerable room for further research in this field, particularly, if we consider incorporating the probabilistic model [53], fuzzy model [95] and inexact model [77] in our general frame work. To include such models, we would need to modify the structure of our models and refine the concepts of minimality, termination, completeness and consistency. Another interesting question is, can these models (**KBS** and **KB**) be extended and used for a nonmonotonic reasoning environment? In a nonmonotonic reasoning situation, such as closed world assumption [67], default logic [68, 69] and other forms of nonmonotonic reasoning [70], the terms "completeness" and "consistency" would have to be redefined; they have different interpretations and meanings. Finally, another natural question is, can these algorithms be parallelized and used for checking the effectiveness of a knowledge base system? All of these considerations provide directions for our future research.

As knowledge base systems find greater use, the need for their effectiveness analysis will grow more important. This dissertation has outlined a framework for the knowledge base effectiveness problem and represents a natural starting point to devise systematic methods to solve such problems. This theoretical work provides a fundamental basis for further development of online automated tools which can be integrated with commercial development environments.

References

1. ''Conference Report : the Alvey Conference,'' *Expert Systems*, vol. 4 no 4, pp. 14 - 15, Manchester UK, July 1987.
2. Anderson, J. R., ''The automated tutoring of introductory computer programming,'' *CACM*, vol. 29, pp. 842 - 849, 1986 .
3. Barr, A. and Feigenbaum, E., in *The Handbook of Artificial Intelligence*, vol. II, William Kaufmann Inc, 1982.
4. Barr, A., Cohen, P. R., and Feigenbaum, E. A., *The Handbook of Artificial Intelligence Volume IV*, Addison-Wesley, 1989.
5. Boehm, B. W., ''A Spiral Model of Software Development and Enhancement,'' *ACM SIGSOFT SOFTWARE ENGINEERING NOTES*, vol. 11 No 4, pp. 14-24, Aug 1986.
6. Bollobas, Bela, *Graph Theory*, Springer-Verlag, New York, 1979.
7. Boose, J. H., ''Personal Construct Theory and the Transfer of Human Expertise,'' *AAAI*, pp. 27-33, 1984.
8. Bramer, M. A., ''A Survey and Critical Review of Expert Systems Research,'' in *Introductory Readings in Expert Systems*, pp. 3 - 29, Gordon and Breach Science Publishers, New York, USA, 1984.
9. Buchanan, B. and Shortliffe, E., ''The Problem of Evaluation,'' in *Rule Based Expert Systems*, ed. B. Buchanan, E. Shortliffe, pp. 571-588, Addison - Wesley, 1985.
10. Casey, T., ''Picking the Right Expert System Application,'' *AI Expert*, pp. 44 - 47, Sept. 1989.
11. Chaudhury, A., Marinescu, D., and Whinston, A., *Net Based Computational Models of Knowledge Processing Systems*, (submitted for publication).

12. Cherniak, C., ''Undebuggability and Cognitive Science,'' *Communications of ACM*, vol. 31 No 4, pp. 402-412, April 1988.

13. Citrenbaum, R., Geissman, J. R., and schultz, R., ''Selecting a Shell,'' *AI EXPERT*, pp. 30 - 39, September 1987.

14. Clancey, W. J., ''Viewing Knowledge Bases as Qualitative Models,'' *IEEE EXPERT*, vol. SUMMER 1989, pp. 9-23.

15. Cohen, P. R., Davis, A., Day, D. S., Greenberg, M., and Kjeldsen, R., ''Representativeness and uncertainty in classification systems,'' *AI Magazine*, vol. 6(3), pp. 136-149, Fall 1985.

16. Cohen, Paul R. and Howe, Adele E., ''Toward AI Research Methodology: Three Case Studies in Evaluation,'' *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 3, pp. 634-646, 1989.

17. Cragun, Brian J and Steudel, Harold J, ''A Decision-Table-Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems,'' *Int. J. Man-Machine Studies*, vol. 26, pp. 633-648, 1987.

18. Davis, R., ''Interactive Transfer of Expertise : Acquisition of New Inference Rules,'' *Artificial Intelligence*, vol. 12 : 2, pp. 121 - 157, North - Holland Publishing Co, 1979.

19. Dreyfus, Hubert L., *What Computers Can't Do: A Critique of Artificial Reason*, Harper & Row, 1972.

20. Dreyfus, Hubert L. and Dreyfus, Stuart L., *MIND OVER MACHINE*, Free Press, 1986.

21. Duda, R., Hart, P., and Nilsson, N., ''Subjective Bayesian Methods for Rule-Based Inference Systems,'' *Proceedings 1976 National Computer Conference*, vol. 45, pp. 1075-1082, AFIPS Press, 1976.

22. Duda, R., Gaschnig, J., and Hart, P., ''Model Design in the Prospector Consultant System for Mineral Exploration,'' in *Expert Systems in the Microelectronic Age*, ed. D. Michie, pp. 153-167, Edinburgh University Press, 1979.

23. Edwards, P., *The Encyclopaedia of Philosophy,* Crowell Collier and Macmillan Inc, New York, 1967.

24. Enderton, H. B., *A Mathematical Introduction to Logic,* Academic Press, New York, 1972.

25. Freedman, R., ''Evaluating Shells: 27-Product Wrap-Up,'' *AI EXPERT,* pp. 69 - 74, September 1987.

26. Frenkel, K. A., ''Complexity and Parallel Processing: An Interview with Richard M. Karp,'' *Communications of ACM*, vol. 29, no 2, pp. 112-117, Feb, 1986.

27. Garey, M. R. and Johnson, D. S., *Computers and Intractability - A Guide to the Theory of NP-Completeness,* W. H. Freeman and Company, New York, 1979.

28. Gaschnig, J., ''Preliminary Performance Analysis of the Prospector Consultant System for Mineral Exploration,'' *Proceeding of Sixth International Joint Conference of Artificial Intelligence*, pp. 308 - 310, Morgan Kaufmann, Los Altos, Calif, 1979.

29. Gaschnig, J., ''Evaluation of Expert Systems: Isuues and Cases Studies,'' in *Building Expert Systems*, ed. F. Hayes-Roth, D. A. Waterman, D. B. Lenat, pp. 241 - 280, Addison-wesley, Reading, Mass, 1983.

30. Genesereth, M. R. and Nilson, N. J., *Foundamental Logic of Artificial Intelligence,* Morgan Kaufman Publishers, Inc, 1987.

31. Ginsberg, A., ''A New Approach to Checking Knowledge Bases For Inconsistency and Redundancy,'' *Third Annual Expert Systems in Goverment Conference*, pp. 102-111, Washington D.C, Oct 19-23, 1987.

32. Ginsberg, A., ''Knowledge-Base Reduction: A New Approach to Checking Knowledge Bases for Inconsistency & Redundancy,'' *Proceeding of AAAI*, vol. II,

pp. 585-589, 1988.

33. Goldberg, A. and Pohl, I., ''Is Complexity Theory of Use to AI,'' in *Artificial & Human Intelligence*, ed. A. Elithorn, R. Banerji, pp. 43 - 56, Elselvier Science Publishers B. V., 1981.

34. Green, C. J. R. and Keyes, M. M., ''Verification and Validation of Expert System,'' *Proceedings of Western Conference on Expert Systems*, pp. 38-45, Anaheim, California, June 2-4. 1987.

35. Hayes-Roth, F., ''Rule-Based Systems,'' *Communications of the ACM*, vol. 28(9), pp. 921 - 932, 1985.

36. Hopcroft, J. E. and Ullman, J. D., *Formal Languages and Their Relation to Automata*, 1976.

37. Howden, W. E., ''The Theory and Practice of Functional Testing,'' *IEEE Software*, vol. 2(5), pp. 6-17, Sep 1985.

38. Howe, P. Cohen, A., ''How Evaluation Guides AI Research,'' *AI Magazine*, vol. 9, No 4, pp. 35-43, Winter 1988.

39. Karp, R. M., ''Combinatorics, Complexity, and Randomness,'' *Communications of ACM*, vol. 29, no. 2, pp. 98-109, Feb, 1986.

40. Klein, P. J. and Dolin, S. B., ''A Problem Features That Influence Design in Expert Systems,'' *Proceedings of Fifth National Conference on Artificial Intelligence*, pp. 956 - 962, 1986.

41. Laufmann, S C., DeVaney, D. Michael, and Whiting, Mark A., ''A Methodology for Evaluating Potential KBS Applications,'' *IEEE EXPERT*, pp. 43 - 62, DECEMBER 1990.

42. Lenat, D. B., Prakash, M., and Shepherd, M., ''CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks,'' *AI Magazine*, vol. 6(4), pp. 65-85, 1986.

43. Lenat, D. B., Guha, R V., Prakash, M., Pittman, D, and Shepherd, M., ''CYC: TOWARD PROGRAMS WITH COMMON SENSE,'' *COMMUNICATION OF ACM*, vol. 33,No 8, pp. 30-49, August, 1990.

44. Levesque, H. J., ''Knowledge Representation and Reasoning,'' in *Annual Review of Computer Science*, vol. 1, pp. 255 - 288, Annual Reviews Inc, 1986.

45. Levi, Keith and Lehner, Paul E., ''Toward an Empirical Approach to Evaluating the Knowledge Base of a Expert System,'' *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, no. 3, pp. 658-662, 1989.

46. Marcot, B., ''Testing Your Knowlegde Base,'' *AI EXPERT*, vol. 2, No 8, pp. 42-47, Aug, 1987.

47. McDermott, J. and Bachant, J., ''R1 Revisited: Four Years in the Trenches,'' *AI Magazine*, vol. 5 No 3, pp. 21 - 32, 1984.

48. Miller, R. A., Pople, H. E., and Myers, J. D., ''Internist-1: An Experimental Computer-Based Diagnostic Consultant for General Internal Medicine,'' *The New England J. Medicine*, vol. 307, No. 8, pp. 468-476, 1982.

49. Morell, L. J., ''Use of Metaknowledge in the Verification of Knowledge-based Systems,'' *Proceedings of The First International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems*, pp. 847-857, Tullahoma, Tennessee, June 1-3, 1988.

50. Nguyen, T., ''Verifying Consistency of Production Systems,'' in *IEEE Proceedings of the Third Conference on AI Applications*, pp. 4 - 8, Washington DC, 1987.

51. Nguyen, T. A., Perkins, W. A., and Laffey, T. J., ''Checking an Expert Systems Knowledge Base for Consistency and Completeness,'' in *Proceedings of the Ninth International Joint Conference on the Artificial Intelligence*, pp. 375 - 378, 1985.

52. Nguyen, Tin A., Perkins, Walton A., Laffey, Thomas J., and Pecora, Deanne, ''Knowledge Base Verification,'' *AI MAGAZINE*, vol. SUMMER, pp. 69 - 75, 1987.

53. Nilsson, N., ''Probability Logic,'' *Artificial Intelligence*, vol. 28, pp. 71-87, 1986.

54. Nilsson, N. J., *Principle of Artificial Intelligence,* Tioga, Palo Alto, Calif, 1980.

55. O'keefe, R. M., Balci, Osman, and Smith, E. P., ''Validating Expert System Performance,'' *IEEE EXPERT,* pp. 81 - 90, Winter 1987.

56. O'Neill, M. and Morris, A., ''Expert Systems in the United Kingdom: an evaluation of development methodologies,'' *Expert Systems*, vol. 6, No. 2, pp. 90-99, April 1989.

57. Ostrand, T. J. and Balcer, M. J., ''The Category-Partition Method for Specifying and Generating Functional Tests,'' *ACM Communication*, vol. 31(6), pp. 676-686, June 1988.

58. Pearl, J., ''Fusion, Propagation, and Structuring in Belief Networks,'' *Artificial Intelligence*, vol. 29, 3, pp. 241-288, Sept 1986.

59. Peterson, J. L., ''Coputation Sequence Sets,'' *Journal of Computer and System Science*, pp. 1 - 24, August, 1976.

60. Peterson, J. L., *Petri Net Theory and the Modeling of Systems,* Prentice-Hall, Inc, 1981.

61. Petri, C., ''Fundamentals of a Theory of Asynchronous Information Flow,'' *Information Processing 62, Proceedings of the 1962 IFIP Congress*, pp. 386 - 390, North-Holland, 1962.

62. Prerau, D. S., ''Selection of an Appropriate Domain for an Expert System,'' *AI MAGAZINE*, vol. 6 (2), pp. 26-50, 1985.

63. Pryor, T. and Gardner, R., ''The HELP system,'' *Information Systems for Patient Care*, pp. 109-128, New York: Springer-Verlag, 1984.

64. Quinlan, J. R., ''Fundamentals of the Knowledge Engineering Problem,'' in *Introductory Readings in Expert Systems*, pp. 33 - 46, Gordon and Breach Science Publishers, New York, USA, 1984.

65. Reddy, R., ``Foundations and Grand Challenges of Artificial Intelligence,'' *AI Magazine*, vol. 9, No 4, pp. 9-21, Winter 1988.

66. Reggia, J. A., Nau, D. S., and Wang, Y., ``A Formal Model of Diagnostic Inference I. Problem Formulation and Decomposition,'' *Inf. Science*, vol. 37, pp. 227-256, 1985.

67. Reiter, R., ``On Closed World Data Bases,'' *Logic and Data Bases*, pp. 55-76, Plenum Press, New York, 1978.

68. Reiter, R., ``A Logic for Default Reasoning,'' *Artificial Intelligence*, vol. 13, pp. 81 - 132, 1980.

69. Reiter, R., ``A Theory of Diagnosis From First Principles,'' *Artificial Intelligence*, vol. 32(1), pp. 57-95, 1987, 1987.

70. Reiter, R., ``Nonmonotonic Reasoning,'' *Ann. Rev. Computer Science*, vol. 2, pp. 147 - 186, 1987.

71. Rich, E. and Waters, R., *Readings in Artificial Intelligence and Software Engineering,* Morgan Kaufmann, Los Altos, Calif, 1986.

72. Rolston, D. W., *Principles of Artificial Intelligence and Expert Systems Development,* McGraw-Hill Book Company, 1988.

73. Rumelhart, D. E. and McClelland, J. L., in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, vol. I, II, MIT Press, Cambridge, Mass, 1986.

74. Schultz, J. R. Geissman, R. D., ``Verification & Validation of Expert Systems,'' *AI expert*, vol. 3, No 2, pp. 26 - 33, Feb 1988.

75. Shortliffe, E., *Computer-Based Medical Consultation : MYCIN,* Elsevier/North - Holland, New York, 1976.

76. Shortliffe, E., Scott, A., Bischoff, M., Campbell, A., Melle, W. Van, and Jacobs, C., ``ONCOCIN: An Expert System for Oncology Protocol Management,'' in

Proceedings of IJCAI 7, pp. 876 - 881, 1981.

77. Shortliffe, E. H. and Buchanan, B. G., ''A Model of Inexact Reasoning in medicine,'' *Mathematical Biosciences*, vol. 23, pp. 351 - 379, 1975.

78. Simon, H. A., ''The Architecture of Complexity,'' *Proceedings of the American Philosophical Society*, vol. 106, no 6, pp. 467-482, Dec, 1962.

79. Slagle, J. R. and Wick, M. R., ''A Method for Evaluating Candidate Expert System Applications,'' *AI MAGAZINE*, pp. 45-53, Winter 1988.

80. Slagle, J R., Gardiner, D A., and Han, K, ''Knowledge Specification of an Expert System,'' *IEEE EXPERT*, vol. 5, pp. 29 - 38, August 1990.

81. Sowa, J. F., *Conceptual Structures: Information Processing in Mind* , Addison-Wesley, Reading, MASS, 1984.

82. Stachowitz, R. and Comb, J., ''Validation of Knowledge-Based Systems,'' in *Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program*, Arlington, VA, 1987.

83. Stachowitz, R. and Comb, J., ''Validation of Expert Systems,'' in *Proceedings of the Twentieth Hawaii International Conference on Systems Sciences*, pp. 686 -695, 1987.

84. Stachowitz, R., ''Building Validation Tools for Knowledge-Based Systems,'' *First Annual Workshop on Space Operations Automation and Robotics (SOAR '87)*, pp. 209 - 216, NASA/JSC, Houston,Texas, Aug 1987.

85. Suwa, M., Scoff, A. C., and Shortliffe, E. H., ''An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System,'' *AI Magazine*, pp. 16 - 21, 1982.

86. Suwa, M., Scott, A., and Shortliffe, E., ''Completeness and Consistency in Rule Based Systems,'' in *Rule Based Expert Systems*, ed. B. Buchanan, E. Shortliffe, pp. 159-170, Addison - Wesley, 1985.

87. Swartout, W. R. and Smoliar, S. W., ''On Making Expert Systems More like Experts,'' *Expert Systems*, vol. 4, No 3, pp. 196-207, Aug, 1987.

88. Ullman, J. D., *Principles of Database and Knowledge-Base Systems,* VOL I, Computer Science Press, Rockvile, MA, 1988.

89. Waterman, D. A., *A Guide to Expert Systems,* Addison-Wesley Publishing Company, 1986.

90. Winston, P., ''Artificial Intelligence: A Perspective,'' in *AI in the 1980s and Beyond*, ed. R. S. Patil, pp. 1-12, MIT Press, 1987.

91. Winston, P. H., *Artificial Intelligence,* Reading Mass, Addison-Wesley, 1984.

92. Winston, P. H., ''The Commercial Debut of Artificial Intelligence,'' in *Applications of Expert Systems*, ed. J. R. Quinlan, pp. 3-22, Turing Institute Press, 1987.

93. Yu, V. L., ''Antimicrobial Selection by a Computer - A Blinded Evaluation by Infectious Disease Experts,'' *Journal of American Medicine Association*, vol. 242, No 12, pp. 1279 - 1282, 1979.

94. Yu, V. L., Fagan, L. M., Bennett, S. W., and Clancey, W. J., ''An Evaluation of MYCIN's Advice,'' in *Rule Based Expert Systems*, ed. B. Buchanan, E. Shortliffe, pp. 589-598, Addison - Wesley, 1985.

95. Zadeh, L. A., ''Syllogistic Reasoning as a Basis for Combinination Of Evidence in Expert Systems,'' *Proc. Int. Joint Conf.Artif. Intell*, pp. 417-419, Los Angeles, 1985.

96. Zhang, D. and Murata, T., ''A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs,'' *IEEE Transactions of Software Engineerin*, vol. 14 No 4, pp. 481 - 497, April 1988.

97. Zhao, Shensheng and Shen, S. N. T., ''On the Consistency Problem of Knowledge Bases,'' *International Conference on Industrial Engineering Applications of Artificial Intelligence and Expert Systems*, pp. 745-750, Charleston, South Carolina, U.S.A, July 15-18, 1990.

AUTOBIOGRAPHY

PLACE AND DATE OF BIRTH:

Shanghai, People's Republic of China, February, 4, 1946

EDUCATION

Ph.D. Candidate Computer Science, Old Dominion University, Norfolk, Virginia, U.S.A., September 1986 - August 1990

M.S. Computer Science, Old Dominion University, Norfolk, Virginia, U.S.A., January 1985 - August 1986

B.S. Physics Science, Beijing University, Beijing, People's Republic of China, September 1963 - September 1969

PUBLICATIONS

1. "Knowledge Representation and Reasoning with Bidirectional Exceptions" (with S N.T. Shen), accepted as journal paper to appear in *International Journal of Modeling and Simulation,* 1990.

2. "On the Consistency Problem of Knowledge Bases" (with S N.T. Shen) the *Thir International Conference on Industrial Engineering Applications of Artificial Intelligence and Expert Systems,* pp 745-750, July 15-18, 1990, at Charleston, South Carolina.

3. "A Tutoring System for Critical Thinking" (with S N.T. Shen and J.Y. Zhang) *Proceedings of the 8th Annual Conf. on Technology and Innovations in Training and Education,* pp 448-458, March 1990, Colorado Springs, Colorado.

4. "On The Completeness Problem of Knowledge Bases" (with S N.T. Shen), *SIXTH IASTED International Conference : EXPERT SYSTEMS THEORY & APPLICATIONS,* pp 95-98, Dec 14 - 16, 1989, Long Beach, California.

5. "Knowledge Representation and Reasoning with Bidirectional Exceptions" (with S N.T. Shen), Proceedings of the *FIFTH IASTED International Symposium : EXPERT SYSTEMS AND NEURAL NETWORKS THEORY & APPLICATIONS,* pp 60-64, August 16 - 18, 1989, Hololulu, Hawaii.

EMPLOYMENT EXPERIENCES

August 1990 - Present : University Professor in Computer Science Dept. at Governors State University, University Park, Illinois, U.S.A.

December 1982 - November 1983 : System software engineer, Beijing Centre for International Economic Information, Beijing, People's Republic of China, responsible for system software maintenance.

May 1981 - December 1982 : Operation manager, Beijing Centre for International Economic Information, Beijing, People's Republic of China, responsible for computer operation and training system operators.

July 1980 - May 1981 : Computer system operator, Beijing Centre for International Economic Information, Beijing, People's Republic of China, responsible for BURROUGHS B6810 large system.

April 1978 - July 1979 : Computer programmer, Beijing Centre for International Economic Information, Beijing, People's Republic of China.

May 1971 - April 1978 : High school mathematics & physics instructor in Hebei, People's Republic of China.