Order Number 8825586

Knowledge acquisition and refinement in expert systems

Tam, Kar Yan, Ph.D.

Purdue University, 1988

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

PLEASE NOTE:

In all cases this material has been filmed in the best possible way from the available copy. Problems encountered with this document have been identified here with a check mark ✓.

1. Glossy photographs or pages _____
2. Colored illustrations, paper or print _____
3. Photographs with dark background _____
4. Illustrations are poor copy _____
5. Pages with black marks, not original copy _____
6. Print shows through as there is text on both sides of page _____
7. Indistinct, broken or small print on several pages ✓
8. Print exceeds margin requirements _____
9. Tightly bound copy with print lost in spine _____
10. Computer printout pages with indistinct print _____
11. Page(s) _____ lacking when material received, and not available from school or author.
12. Page(s) _____ seem to be missing in numbering only as text follows.
13. Two pages numbered _____. Text follows.
14. Curling and wrinkled pages _____
15. Dissertation contains pages with print at a slant, filmed as received _____
16. Other _____

_____

_____

KNOWLEDGE ACQUISITION AND REFINEMENT IN EXPERT SYSTEMS

A Thesis
Submitted to the Faculty

of

Purdue University

by

Kar Yan Tam

In Partial Fulfillment of the
Requirements for the Degree

of

Doctor of Philosophy

May 1988

PURDUE UNIVERSITY

Graduate School

This is to certify that the thesis prepared

By Tam, Kar Yan

Entitled

Knowledge Acquisition and Refinement of Expert Systems

Complies with University regulations and meets the standards of the Graduate School for originality and quality

For the degree of Doctor of Philosophy

Signed by the final examining committee:

Andrew Whinston, chair

Leah H Jamieson

Clyde W Holsapple

Sheng Cheng Liu

Approved by the head of school or department:

March 28 1988 Donald C King

☐ is
This thesis ☒ is not to be regarded as confidential

Andrew Whinston
Major professor

Grad. School
Form No. 9
Revised 9-85

ACKNOWLEDGMENTS

I would like to thank Professors Andrew B. Whinston and Clyde W. Holsapple, for the many hours they spent with me discussing various aspects of the research reported in this thesis.

The financial support of IBM during my first two years at Krannert and the Purdue Research Foundation for the David Ross Dissertation Grant are also gratefully acknowledged.

TABLE OF CONTENTS

Page

Page

LIST OF FIGURES

Figure Page

ABSTRACT

Tam, Kar Yan. Ph.D., Purdue University, May 1988. Knowledge Acquisition and Refinement in Expert Systems. Major Professor : Andrew B. Whinston.

The issue of knowledge refinement in expert systems is addressed in this thesis. In general, an expert system is composed of a knowledge base which stores application specific reasoning knowledge, an inference engine which processes the stored knowledge, and an interface through which communication links between the users and the expert system are established. In terms of knowledge refinement, this architecture is dependent on knowledge engineers to refine its stored knowledge on a periodical basis. The frequency with which the knowledge base is revised depends very much on the underlying application domain. Furthermore, the control mechanism of the inference engine may also need to be updated in order to match up with the changing inference process of human experts. In the scope of this thesis, we will primarily focus on the former.

In this thesis, we will generalize the principle of knowledge acquisition to knowledge refinement of a continuous nature. While knowledge acquisition takes place in the early stage of an expert system development,

knowledge refinement is applicable during the entire life-span of an expert system. The thesis starts by presenting a conceptual framework of building knowledge acquisition systems. Based on this framework, a generic architecture of expert systems with a provision to refine its own knowledge base is discussed.

The novelty of this research is to study knowledge acquisition and refinement in expert system by presenting an architecture and to prove its validity, at least partially, by streamlining it to some generic problem tasks which are illustrated with a real-life application.

The thesis is organized as follows : Chapter 1 presents a conceptual framework of building knowledge acquisition systems. The idea of knowledge acquisition is extended to knowledge refinement in Chapter 2; a generic architecture of expert system that are capable of self-refinement of knowledge is presented. The architecture is then used to construct expert systems to perform generic problem tasks of pattern recognition and classifiaction in Chapter 3 and Chapter 4 respectively. Chapter 5 concludes the thesis by discussing future research directions of knowledge refinement.

CHAPTER 1
A CONCEPTUAL FRAMEWORK OF BUILDING KNOWLEDGE ACQUISITION SYSTEMS

1.1. Introduction

The process of knowledge acquisition is generally regarded as the bottleneck in developing expert systems [Buchanan 1982]. In a broader context, knowledge acquisition refers to the collective process of eliciting, structuring and coding of human expertise in a form that is computationally feasible on a computer.

Traditionally the task is accomplished through an iterative process of interviewing between a knowledge engineer and an expert. The interrogating process, though commonly adopted by knowledge engineers, has a number of pitfalls from a cognitive standpoint. The major obstacles against this approach are the inabilities of knowledge engineers to ask "to the point" questions and the inabilities of experts to articulate their expertise in concrete terms [Bainbridge 1987] .

Eliciting knowledge by asking questions requires a knowledge engineer to has at least a certain degree of acquaintance with the problem

domain. For instance in medical diagnosis, the concepts such as "symptoms" and "diseases" have to be fully understood beforehand. Otherwise, the validity of the knowledge base will very likely be impaired. In some domains, the background knowledge required is enormous, making the interview technique infeasible and inaccurate if applied.

Furthermore, it is not uncommon that experts make their decisions which are not available for conscious introspection. Cognitively, this phenomenon occurs because human often cannot get access to their mental processes that lead to their final decisions. Ericsson and Simon [1984] suggested that only those information reside in short term memory can be verbalized. Since expert knowledge embrace skills, experiences, and problem solving techniques that have evolved through years of practice, they are particularly difficult to articulate using the interview technique.

Various other approaches have been proposed to circumvent the bottleneck problem in knowledge acquisition. According to the role of computer played in the process of knowledge acquisition and the degree of learning performed on the part of the acquirer (a knowledge engineer or a computer based knowledge acquisition system), they can be classified into two main streams of approaches.

The first approach follows the direction of interviewing by improving the skills and introducing new techniques to aid a knowledge engineer in the knowledge acquisition process. The setting of this approach is still an iterative process between two parties - knowledge engineer and expert.

Newell and Simon [1972] proposed a way to elicit human expertise by having the experts to "think aloud" in a problem solving process. This method is called Protocol Analysis and is suggested to be a methodology in developing expert systems [Waterman 1971]. The idea is to study how experts solve problems by recording verbal transcripts of sample problem solving processes. These transcripts are then analyzed in detail. A transcript represents a solution path in solving a problem. It reveals sequence of problem solving events conducted by an expert in a problem solving episode.

A more elaborate variant of protocol analysis is taken in [Belkin 1986] in which interactions between users and librarians in document retrieval situations are first observed and recorded in the form of audio transcripts. Next, discourse analysis [Hendrix 1979] is applied to the transcripts to identify and specify the functions of an intelligent interface for a document retrieval system.

For protocol analysis, the knowledge granularity is at the utterance level, making the analysis a very time consuming process. Furthermore, analysts require considerable training in psychology in order to attain a satisfactory competence level. Ecrisson and Simon in [1984] gave a comprehensive account on the problems of eliciting knowledge using verbal data.

Another approach is taken in [Lafrance 1986] to formalize the interviewing process by proposing a knowledge acquisition grid. A knowledge aquisition grid represents a taxonomy of question types and forms of knowledge. It serves as a framework and a systematic way to train knowledge engineers to relate the form of knowledge (eg. Scripts, Rules-of Thumb) with the types of questions (eg. cataloging categories, ascertaining attributes) to be asked during an interview.

One common feature shared by these techniques is that they are basically manual techniques. The use of computer is limited to the analysis of data (eg. utterances in a transcript) rather than on the acquisition process itself. As a consequence, the inherent limits associated with these techniques on the effectiveness, completeness and cost of the process have led to research in automating the task by building computer based knowledge acquisition systems. By delegating the

task to a computer, the duration of the process can be shortened to a large extent by :

1) Direct encoding of expertise in a form recognizable by the inference engine.

2) Allowing rapid construction of prototypes to be assessed by experts and end users in the early stage of the development process.

3) Reducing the time spent in debugging the knowledge base by avoiding the possibility of an inconsistent knowledge base. This is accomplished by checking the consistency and completeness of the acquired expertise using a deductive mechanism.

In this chapter, the issue of building knowledge acquisition systems is addressed from the perspective of machine learning. We will present a framework that identifies the essential attributes of a knowledge acquisition system. The objective is twofolded. First, it explores the various techniques used in knowledge acquisition. Second, a framework is presented to provide a systematic view and guidelines for the design of knowledge acquisition systems. Attention is placed on the mapping between learning strategies and the other attributes of a knowledge acquisition system. By doing this, we attempt to abstract out the crucial decisions pertaining to the design of a knowledge acquisition system. In

Section 1.2., the various learning strategies are discussed. Section 1.3. presents a framework that identifies the attributes of a knowledge acquisition system and discusses how these attributes are related to the various learning strategies. Section 1.4. concludes the chapter with a discussion on the implications of the framework.

1.2. Learning Strategies

Learning is defined in a number of ways. In [Simon 83], Simon defined learning as

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time."

It is plausible to equate our connotation of "learning" with "knowledge acquisition" in the context of building systems that facilitate the acquisition of expertise. Indeed, numerous knowledge acquisition systems actually learn the domain knowledge from the experts [Cohen and Feigenbaum 1982]. They perform more than the transfer of expertise between media but act as individual entities that, like human, enrich their

knowledge base through learning. One ambitious goal of Artificial Intelligence is to build expert systems that are capable to learn on their own. To this end, the theory of learning, especially machine learning [Simon 1983],[Carbonell, Michakski, and Mitchell 1983], provides valuable conceptual foundation and insights to research in this area.

Given our interest in building knowledge acquisition systems, we are more interested in the learning process itself. Unlike the definitions of learning which are numerous, there are basically four different approaches towards the process of learning : rote learning, learning by deduction, learning by analogy, and learning by induction. Notice that the complexity of the process and the degree of inference on the part of the learner increase in this order. In brief, a learning strategy describes how "new" knowledge is acquired and synthesized by the learner. The various strategies are discussed below.

1.2.1. Rote Learning

Rote learning refers to the direct implantation of knowledge. It involves two parties : a student and a teacher. Knowledge is spoon-feed into the student by the teacher . The role of the student is passive and the degree of inference on the part of a student is very limited during the

learning process. In its strictest form, no modification or rejection of the implanted knowledge takes place during the process. The major activity conducted during the acquisition process is to index the acquired knowledge for later retrieval. Programming is a form of rote learning in the sense that instructions are provided by the programmer, and the computer follows whatever it is told to do.

Obviously, the ways knowledge engineers acquire knowledge do not fall into this category. However in cases where the knowledge engineer himself is an expert in the application area, rote learning is the most direct and effective approach to build expert systems. The long iterative process of interviewing can then be eliminated. The knowledge engineer simply transplants his own expertise to a computer in a similar fashion as a programmer writes a program. This is aided by declarative languages such as Prolog which allows an expert to state the problem and its solution method in direct declarative form. In fact, a number of successful expert systems are built by professionals in their own areas.

1.2.2. Learning by Deduction

Added to the core knowledge acquired with a set of inference rules results in a strategy called learning by deduction. The inference rules

specify how additional knowledge can be deduced from the core knowledge during the learning process. No "new" knowledge will be created using this method. This is because the scope of the knowledge will be defined once the core knowledge and the set of inference rules are specified. The inference rules and the initial set of knowledge are provided by the teacher. Learning a piece of information involves the invocation of a deduction mechanism which is computationally more complicate than the indexing structure used in rote learning. One can consider the choice between rote learning and deductive learning as one driven by the tradeoff between space and time. In logical terms, for instance, the fact that all mammals are animals can be stated as : For all x, Mammal(x) => Animal(x). In rote learning, to store the fact that human beings are both mammal and animial necessitates both Mammal(human) and Animal (human) to be physically resided in the knowledge base. In deductive learning systems, only the fact Mammal(human) and the above statement needed to be stored. The fact Animal(human) can be deduced by applying the modus ponen inference rule as follows :

Mammal(human)

For all x, Mammal(x) => Animal(x)

Animal(human)

The price to pay for the space gained in deductive systems is the increase in time spent in performing the deduction as illustrated by the above example.

1.2.3. Learning by Analogy

To learn by analogy is to create new concepts by transforming and augmenting existing ones which are similar to the new concepts. A concept can be a physical object or a problem solving method. The process is usually broken down into two phases. In the first phase, existing concepts which bear strong similarity with the new concept are searched for. These concepts are then mapped to the new concept during the second phase. In order to be successful, this approach requires a measure of similarity. A general definition of this measure is difficult. It depends on the kinds of concept that we are considering and the context of comparison.

Learning by analogy requires a certain amount of inference on the part of the learner in the way that a learner can choose the similarity measure that he will use to acquire knowledge. Suppose a learner thinks that driving a tank is similar to driving a bulldozer, the direction he will pursue in learning the concept of a tank will be very different from the one that he thinks buses and tanks are similar.

1.2.4. Learning by Induction

Induction is to infer from specific to general. Usually, the induction process is driven by a model that specifies the form of relationship that we want to learn from a collection of examples. Examples of a concept are analyzed and the results are used to fill in the details of the model. The role of the teacher is to provide advices in the selection of models and the criteria of the induction process.

The discipline of statistics is the study of inductive inference from a rigorous mathematical perspective. For instance, a linear regression model assumes a linear relationship between the given dependent and independent variables. The induction process is to infer a linear regression line that best matches the examples. The goodness of fit measure is also defined in the model (ie. least square) and the actual value of which depends on the examples in hand.

In acquiring diagnosis rules in medicine, a knowledge engineer might try to infer symptoms-disease rules from past diagnosis cases. In essence, he tries to generalize these diagonses so that they can be applied in future cases. Notice that the induction process is examples driven and is subject to bias data samples. Also, the learning process depends on the number and types of examples (ie. positive and negative) available.

1.2.5. Factors Affecting the Choice of Learning Strategies

Except for some ad hoc systems, almost all existing learning systems fall into one of these categories or their variants. The question of which one to apply in building a knowledge acquisition system depends on a number of factors as follows :

1) **Problem domain** - The new problem domain and its relationships with others have significant impacts on the efficiency and effectiveness of the knowledge acquisition process. Three kinds of relationships are identified .

a) **Disjoint problem domains** - Two domains are disjoint when they are very different in their problem contexts. In these situations, there is no apparent learning strategy that is preferred over others. But in cases when two domains are identified to be very similar, analogy learning technique can be applied to learn the new concept from existing ones.

b) **Problem domain is covered by other existing domains** - If a problem domain is known to be covered by others that have already been learned, then the new domain can be learned by imposing conditions on the covering domain. The covering relationships between concepts can be represented by a hierarchy tree. By climbing down the hierarchy tree, the knowledge that are common between the two can then be deduced using

the background knowledge provided by the more general one.For instance, the concept Animal properly contains the concept of Mammal. Therefore, by imposing conditions on Animal such as

Is-a (x, Mammal) => Is-a (x, Animal), and

Feed-milk (x) => Is-a (x, Mammal),

we can deduce all properties of Mammal from Animals.

c) **Problem domain covers other domains** - On the other hand, if the new domain covers some existing domains, then the new domain can be learned by generalizing the existing ones. Unlike climbing down the hierarchy tree which is truth preserving, climbing up the hierarchy tree is false preserving. This is because the generalization process, which is basically an induction procedure, does not guarantee to produce knowledge that are valid for the new domain.

2) **Problem type** - Problems, despite their domains, can be classified into generic tasks (eg. planning, classification) as proposed in [Chandrasekaran 1986]. In fact, techniques have been developed to acquire knowledge for some generic tasks. These techniques might adopt different learning strategies. Thus, given a problem that can be classified into a generic task, existing learning strategies for this generic task can be applied.

3) **Availability of expertise** - In cases when there is an abundance of expertise or the cost of expertise is low, it is feasible to adopt less complicate learning strategies such as rote learning and deductive learning. However, in situations where expertise is scarce and costly, a learner has to play a more active role in inferring the knowledge from the limited knowledge sources. Learning by analogy and induction are resorted to in these situations by transferring the load from the experts to the learners.

In order to carry out the task of knowledge acquisition in an effective manner, a system should not be restricted to a single learning strategy. Indeed, existing systems usually exhibit hybrid learning behavior.

1.3. A Framework of Knowledge Acquisition Systems

Fig. 1.1. depicts a framework of knowledge acquisition systems in the form of a grid. The vertical dimension represents the various learning strategies discussed in the previous section. To implement a learning strategy (or combination of strategies) chosen along this dimension, the designer of a knowledge acquisition system has to decide on the issues shown on the horizontal dimension - knowledge representation and elicitation, learning operator, and learning criteria. Together with the

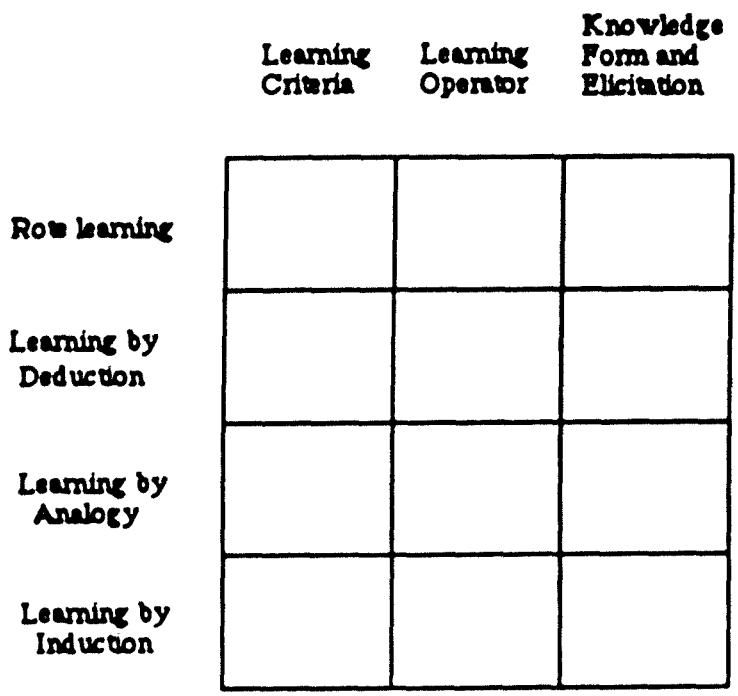

Fig. 1.1. A Framework of Knowledge Acquisition Systems

learning strategy, these correspond to the essential attributes of a knowledge acquisition system. Our framework does not intend to serve the purpose of a taxonomy for knowledge acquisition systems. Yet, it attempts to assist the design of these systems by identifying the essential features and the crucial design issues.

1.3.1. Knowledge Representation and Elicitation

The issue of knowledge representation is concerned with the specification of a scheme to state concepts and their relationships. A number of knowledge representation schema have been proposed [Chang 1973], [Quillian 1968], [Minsky 1975]. Some are designed for specific application [Shortliffe 1976] whereas others are for common sense reasoning [Schank 1975], [Minsky 1975], [McCarthy 1968], [Raphael 1968]. One question encountered in designing knowledge acquisition systems is how the final form of knowledge can affect the knowledge acquisition process ? Answer to this question is crucial to the selection of a knowledge acquisition methodology. The question can be broken down into three more refined questions - 1) What are the different forms of knowledge ? 2) What are the criteria in selecting knowledge representation forms ? 3) How to elicit knowledge from experts ? Here, a distinction is

drawn between the grand process of knowledge acquisition and the more specific task of knowledge elicitation. The latter is merely concerned with the articulation of expertise and the provision of tools to facilitate it.

1.3.1.1. Knowledge Forms and Selection Criteria

The notion of generic tasks suggested by Chandrasekaran [1983], [1984], [1986] provides a direction in addressing the first two questions. In [Chandrasekaran 1986], a generic task is defined as "basic combinations of knowledge and inference strategies that are powerful for dealing for certain kinds of tasks". For instance, medical diagnosis and fault detection can both be categorized as classification tasks. Both of them involved mapping data (symptoms and defectives) to decisions (diseases and faults). The two problems, though different in their domain areas, are structurally similar in their solution methods - diagnosis rules. Based on the notion of a generic task, it is natural to have different generic knowledge acquisition methodologies associated with different generic tasks. The mapping is shown in Fig. 1. 2.

In view of existing systems, there seems to exist evidences to support the above problem-task-knowledge-acquisition mapping. Here, we focus on the latter part of the mapping (i.e. between knowledge forms and

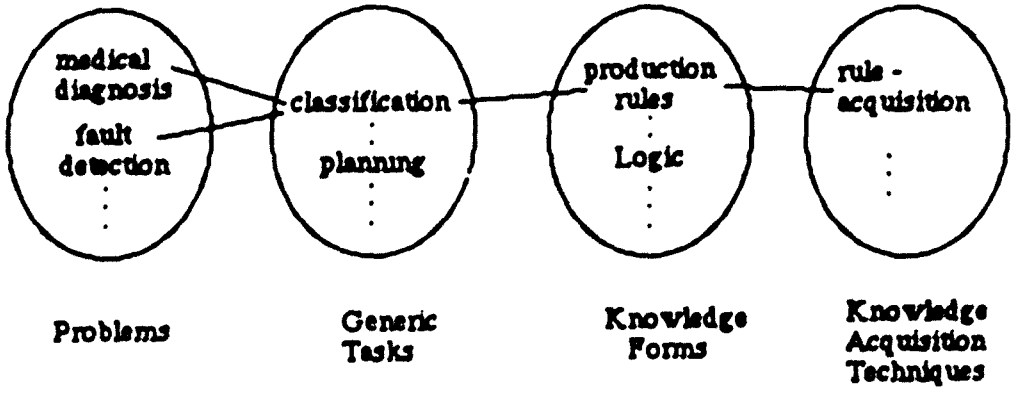

Fig. 1.2. Problem-Task-Form-Acquisition Mapping

acquisition techniques). Let us briefly review common classes of knowledge representation and their learning techniques.

Logic - Mechanical theorem proving [Chang 1973] as the knowledge processing technique is employed by expert systems using First Order Logic (FOL). Facts and relationships are stated as logical statements. A logical statement is a collection of primitive assertions (ie. predicates) connected by logical operators (eg. AND, OR, NOT). Learning techniques have been developed to acquire knowledge stated in FOL or the likes. Sammut described a system MARVIN [Sammut 1981] that can learn a

concept stated in Prolog like statement by generating instances of the concept (eg. the geometric configuration of an arch) and check with the expert whether they are consistent or not. If an instance is too general, it will be specified until it is consistent with the expert's conclusion. In [Buntine 1986], Buntine discussed an induction algorithm for Horn Clauses. Given a collection of Horn clauses, the algorithm infers relationships between these clauses by generalizing the initial set of clauses . PLANT/DS, a system developed by Michaski and Chilausky [1980] to acquire soybean disease diagnosis rules, used an extend version

of FOL called APC - Annotated Predicate Calculus.In these systems, a logical statement is a description of a concept or an object. Interestingly, induction is the popular strategy used to learn a concept. Examples of a concept are stated in terms of logical statements. The generic approach is to generalize these statements to ones that are more descriptive and at the same time consistent with the given examples. More will be discussed on learning operators in the next section.

Semantic Network/Frame - Marker propagation is used in semantic network based knowledge systems in problem solving. Using semantic network, concepts are represented in the form of graphs with nodes representing instances, concepts (Color) and attributes (eg. Blue), and arcs representing relationships (eg. is-a) Frame is a restricted form of semantic network [Minsky 1975] which has been used for common sense and default reasoning. This is made possible by the default slots of a frame. By assuming default values for these slots unless otherwise specified, generic schema for classes of problem solving methods can be represented. In [Winston 1975], Winston presented a learning program that learned the structural descriptions of objects in the block world using induction technique. Examples of an object are provided. These examples

take the form of graphs. Differences and similarities between examples are derived by matching pairs of graphs. Three kinds of match for both nodes and links are defined : completely match, partially match, and does-not match. Information obtained is used to control the search of a general description of the concept.

Production Rules - Knowledge in the form of IF-THEN rules is another common knowledge representation scheme. Production rule is similar to logic in its form of representation. The condition and conclusion of a rule can be readily translated into two conjunctive statements with the former implying the latter. Thus, learning techniques associated with logic can be applied in rule based systems as well. However, there are basic differences between the two in their inference mechanisms. Instead of having only one inference mechanism (resolution-refutation), different inference procedures are adopted in different production systems. For instance, backward chaining is used in MYCIN [Shortliffe 1976] while a recognize-act cycle is adopted in OPS5 [Forgy 1981] as the inference mechanism. Intelligent knowledge acquisiton systems called rule acquisition systems such as RULEGEN in meta-DENDRAL [Buchanan

and Mitchell 1978] and AQ11 [Michalski and Larson 1978] have been built to acquire rules using the induction approach.

Although not completed, the forms of knowledge discussed above cover a large portion of existing expert systems which associated with a wide range of generic tasks (eg. script, classification, planning). The notion of classifying problems into generic tasks enables us to select a knowledge form which is the most appropriate for a given problem. Once a knowledge form is chosen, the acquisition technique follows accordingly. Thus, it seems plausible to design generic acquisition procedures for generic tasks. In search of a classification scheme for generic tasks, however, problems remain in deciding the set of criteria of the classification scheme. Nevertheless, it provides a framework to guide knowledge engineers to relate task domains to knowledge forms and to the selection of acquisition techniques.

1.3.1.2. Knowledge Elicitation - A Psychological Approach

Once a knowledge form is selected, the next step is the eliciting and coding of expertise. The encoded knowledge might not be in its final form. This initial chunk of knowledge provides the basis for further refinement

and modification. In learning by induciton, this corresponds to the examples that will be generalized by the induction procedure.

As mentioned in Section 1.1., expert knowledge are difficult to elicit in situations when they are not subject to conscious introspection. Expert knowledge usually contain context senstive concepts which are difficult to articulate in isolation from others or without a proper context. For instance, an expert in political affairs might not accurately express his assessments on current U.S./U.S.S.R relation without referring to previous events. Furthermore, it is easier for the expert to focus on certain traits of a concept at a time rather than to come up with grand statements describing it. This is also true in making comparisons between concepts. Comparing pairs of concepts are more informative than between a concept and a group of concepts. Despite the advantage of fragmenting the knowledge domain, problems remain as how to synthesize back the individual knowledge fragements.

There is an emerging role of Psychology in this endeavor. Numerous knowledge acquisition systems have been built using techniques from psychology. These techniques, embracing personal construct theory, multidimensional scaling, and clustering, are generally called measurement and scaling techniques.

These techniques provide methodologies to measure, compare, and categorize fuzzy concepts, especially perceptions and physical feelings which are difficult to articulate by experts in absolute terms. In medical diagnosis where the feeling of a patient is an important factor in determining a diagnostic action, these measurement techniques have been valuable tools to elicit this information. They are suggested in [Gammack & Young 1984] as means of eliciting knowledge form experts for the purpose of building expert systems. Let us briefly review these techniques.

Personal Construct - The personal construct theory proposed by Kelly [1955] suggested that each individual seeks to predict and control events by creating theories of the world. Furthermore, these theories exist in the form of constructs in each person's mind. Kelly defined a construct as a bipolar scaled dimension measuring the similarity and contrast between events. Personal construct theory has been the basis for systems such as ETS [Boose 1984], [Boose 1985] and AQUINAS [Boose and Bradshaw 1986]. Personal constructs are implemented in the form of repertory grids in these systems. In its simplest form, a grid consists of two dimensions - concepts and traits. Entries of a grid are numbers with values

range from 1 to 5. Each number corresponds to a relative measure of a trait associated with a concept. To elicit knowledge from experts using this technique includes the followings : 1) identify the concepts, 2) identify the traits that discriminate the concepts in 1), and 3) fill in the entries of the grid form by 1) and 2).

Multidimensional scaling - Multidimensional scaling technique is a variant of least square fitting methods. The input data is a symmetric matrix with each element representing the distance between two concepts. Depending on the subject to be studied, the distance can be a measure of similarity or difference. In the context of medical diagnosis, entries of the distance matrix, for instance, might represent the relative degree of pain between pairs of heart diseases. The iterative algorithm then attempts to fit the distance matrix to the required dimensions by minimizing the stress [Kruskal 1964]. Plots showing the relative positions of concepts can then be obtained for different pairs of dimensions. The meaning of each dimension is still subjected to human interpretation. For the purpose of eliciting knowledge, the technique can be used in both direction. In the reverse direction, this is done by presenting to an expert a plot of concepts

and ask the expert to locate the appropriate position of the new concept on the map.

Clustering - Clustering technique is generally used as a taxonomical tool [Johnson 1967]. Input to the cluster procedure is a symmetric distance matrix and a number specifying the number of clusters to be formed. The dimensions of the matrix is the same as the the number of data objects. There are various types of clustering procedure with hierarchical clustering procedures being the most common one. In an iterative fashion, a hierarchical procedure searches and merges the closest pair of objects into one object until the number of objects is reduced to the required number. The output is a taxonomy of the objects in the form of a tree.

1.3.2. Learning Operators

During the learning process, knowledge is constantly being revised or generated from existing ones. Learning operators are the means to alter the descriptions of a concept. These operators are actually functions that map a concept to another. Conceptually, there is no learning operator in a learning system based on rote learning because the student is not allowed to change the description of a concept once told by the

teacher. However, for retrieval purposes, the form of representation might be changed, allowing an effective indexing structure to be constructed.

In case of a deductive learning system, learning operators correspond to the inference rules associated with the system. First Order Logic provides a formal model for many deductive learning systems. It provides a language to state knowledge and rules of inferences to make deductions. The knowledge so generated is not "new" but logical consequence of a deduction process. Modus Ponen together with universal generalization and existential generalization form the core of inferences rule in First Order Logic.

To learn by analogy normally requires the determination of a sequence of operations necessary to map a concept to another. The mapping process is the computational procedure associated with an analogy measure. Using the notion of problem space introduced by Newell and Simon [1972], Carbonell [1983] defined analogy learning as a process of mapping and searching in two spaces - the original problem space and the analogy transform problem space. A solution to a problem represents a path in the problem space. To solve a similar problem using the analogical approach, existing solution of a similar problem is

mapped into the analogy transform space as a point representing the initial state of the transform space. A series of transformation is applied to the initial state until a new state in the transform space that satisfies the specification of the new problem is obtained. The new state in the transform space is then mapped back to the original problem space as a solution path for the new problem. Under Carbonell's framework, the transformation operators are the learning operators that perform the mapping process.

In learning by induction, despite the form of knowledge selected, there are basically two kinds of operators for systems using induction techniques, namely, generalization and specialization operators. For example, the concept Person(John) can be generalized to for all x, Person(x). On the other hand, by imposing more condition to the concept, it can be specialized to

For all x, Student(x) => Person(x)

That is, John is a person only if he is also a student. These two kinds of operators vary in their operations under different knowledge forms. Yet, they all serve the functions of

1) generalizing a concept

2) specializing a concept

In [Michalski 1983], these operators take the form of rewriting rules. Some of the essential operators are listed below :

Generalization Operators	Specialization Operators
1. Dropping condition	Adding condition
2. Adding alternative	Dropping alternative
3. Extending reference	Closing reference
4. Climbing generalization tree	Climbing specialization tree

In [Winston 1975] semantic network is used to represent structural objects and their relationships. Examples of a concept such as an arch is represented by graphs. Generalizations and specializations of concepts take place by evoking graph operations - deleting, adding, and comparing links and nodes.

The selection of learning operators depends on 1) the learning strategy used and 2) the knowledge representation form. Therefore, it is essential to define these two attributes of a knowledge acquisition system before the issue of learning operators can be addressed.

1.3.3. Learning Criteria

Learning criteria determine the goal of a learning process. Some of these criteria might be in conflict with one another and a compromise is usually required to resolve this. In rote learning systems, the obvious criterion is that the knowledge acquired is identical to the ones supplied by the teacher. The student simply makes sure that all instructions are received and properly indexed.

In the case of deductive systems, the basic criteria are consistency and completeness. In logical terms, consistency means, given an interpretation, all true statements can be deducted using the inference rules. Completeness refers to the ability to find a contradiction if one exists. Ideally in learning by deduction, new knowledge acquired from experts or generated by the system itself should satisfy these two criteria. Yet, the two criteria are seldom satisfied in practice deal to limited computation resources.

In systems that learn by analogy, learning criterion is based on the similarity measure defined by the system. The analogy measure is expressed as a function that map two concepts to a real number as follow:

Analogy : Dom(Concept)XDom(Concept) --> R,

where Dom(Concept) denotes the domain of a concept.

The simplest example of an analogy is the function absolute difference (ie. ||) of two real numbers. Given the definition is a topological one, for example, 3 is more closer to 4 than 5. That is, |3-4| < |3-5|. For concepts that are multidimensional - one that can be described by a set of features. Tversky [1977] presented a similarity measure which defined on the feature set of a concept. The degree of similarity between two concepts increases with the size of the intersection of their sets and decreases with the size of the intersection of the two complement sets. For concepts that are identified by their structures, the measure of analogy is defined on the sequence of structural transformations required to transform a concept to another. Each transformation is given a weight (or cost). By adding up all the weights associated with a transformation sequence, one obtains an index indicating the similarity between the two concepts. One example is the measuring of distance of two sequences of symbols [Levenshtein 1966]. These measures are concerned with the apparent structural differences between two concepts.

For more complicate knowledge structures, Gentner in [1980], [1983] described a mapping process that transform a descriptive structure

from one domain (base) to another (target). Predicates and relations are systematically deleted and added to the base structure to match with the target structure. The theory of structural mapping is the basis of CARL [Burstein 1983]. CARL is a learning system designed to learn the semantics of assignment statements for the BASIC programming language. Concepts such as PUT-IN-BOX are stated in frame like structures. During the learning process, CARL draws on analogy between variables in a computer and objects in a box. The success of analogical learning depends heavily on the choice of the analogy measure [Winston 1980]. However , a general measure of analogy is difficult to determine. This is especially true when the meaning of analogy between two concepts varies under different perceptions and situations.

Angluin and Smith [1983] surveyed different induction methods and criteria from both the practical and theoretical standpoints. In general, there are two fundamental criteria pertaining to learning by induction namely, simplicity and goodness of fit.

Simplicity - In learning a concept, a simple yet powerful description is most desired. Simplicity can be interpreted as the general applicability of a concept description. A very specific description might find itself too

limited to be applied in different settings. Furthermore, human tend to form stereotype of concepts for the purposes of efficient storage and prediction. An induction procedure, which is designed to replace knowledge engineers in acquiring expert knowledge, should consider this cognitive behavior as a primary criterion.

Goodness of fit - A simple description, however, might be too broad to apply in other situations. In order to be useful, specific instances of a concept cannot be generalized too much. To restrict the degree of generalization, examples of a concept is provided to guide the generalization process in such a way that the final concept description obtained should be consistent with the examples provided.

These two criteria exist in various forms in different learning systems. In [Larson and Michalski 1977],[Hayes-Roth 1976], they take the form of maximally-specific conjunctive generalizations (MSC-generalizations). A conjunctive generalization is a description of a concept obtained by forming the conjunction of a group of primitive statements (eg. predicates). A maximally-specific conjunctive

generalization is the most detailed description that holds true for of all examples of the concept.

However, MSC-generalization is primarily used to generate description for a set of positive examples. Only positive traits of the examples are taken into account, making the MSC-generalization too general to apply in other circumstances. To overcome this, negative examples of a concept are also used to restrict the degree of generalization. When negative examples are used, the criteria take the form of two parameters which specify the number of positive and negative examples covered and rejected by a rule [Tam, Holsapple and Whinston 1987]. A special kind of negative examples called Near-Miss examples is suggested by Winston [1975] to control the generalization process. A Near-Miss example is a negative example that differs in one attribute from the concept. Near-Miss examples offer valuable information in identifying a concept. Yet in general, Near-Miss examples are difficult to obtain.

1.4. Conclusion

In this chapter, we have presented a framework for building knowledge acquisition systems. The interplay between learning strategy,

knowledge form, learning operator, learning criteria, and their influencing factors are summarized in Fig. 1.3. Each circle represents an attribute which has to be determined during the design process. Squares denote factors affecting the decisions pertaining to an attribute. Arrows representing dependence relationship. Implications derived from

Fig. 1.3. can serve as guidelines in designing a knowledge acquisition system by

1) identifying the essential attributes of a knowledge acquisition system - We have identified four attributes : learning strategy, knowledge representation and elicitation, learning operator, and learning criteria. attributes.

2) identifying the intimate relationships between these attributes and the factors pertaining to the application domain - There are a number of factors affecting the choice of each attribute. As shown in Fig. 1.3., the choice of a learning strategy depends on three factors : problem domain, problem type, and availability of expertise. The notion of generic tasks enables us to select a proper knowledge form by classifying problems into generic tasks and to map these tasks to their representation forms. For knowledge that is difficult to elicit, measurement and scaling techniques

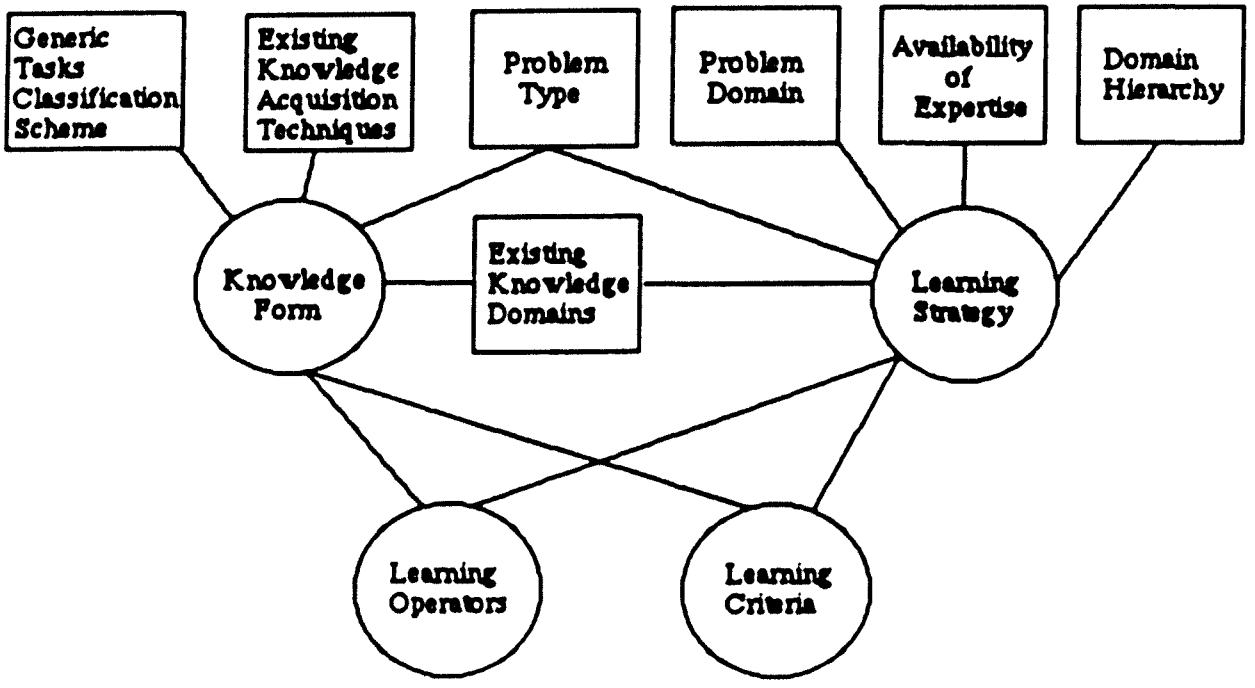

Fig. 1.3. Dependence Relationship between the Four Attributes and their Influencing Factors

are suggested. As soon as a learning strategy (or combination of strategies) and a knowledge form is determined, the definitions of learning operators and learning criteria follow accordingly.

3) providing a framework for the design procedure - The following outlines a design procedure as implied by Fig. 1.3. First, the problem domain and types are identified. Next , the new problem domain is compared with existing knowledge domains to discover the various relationships (see Section 1.2.). Information obtained is used to choose a learning strategy. Given a generic task classification scheme, a proper knowledge form is selected. At this point, both the attributes of knowledge representation and learning strategy is defined. The next step is to elicit expertise and encoded them in the knowledge form selected. To elicit and ecode this knowledge might require application of phychological techniques in situations where expertise is difficult to articulate. Finally, the issue of learning operators and learning criteria are defined. As mentioned in Section 1.3., these two attributes depends on the learning strategies selected.

CHAPTER 2

KNOWLEDGE REFINEMENT OF EXPERT SYSTEMS

2.1. Introduction

The conventional way of organizing an expert system as depicted in Fig.2.1. does not provide adequate interfacing between the knowledge acquisition system and the other components of an expert system. Knowledge acquisition and refinement are mostly manual tasks that are performed exogeneous of the expert system. The revised knowledge is then transplanted back to the knowledge base. The process is laborious and renders it very inefficient and uneconomical to apply expert systems in rapid changing domains.

In general, the term "knowledge acquisition" refers to the process of collecting domain specific knowledge from human experts. The various techniques to acquire knowledge have been discussed in the previous chapter While knowledge acquisition is a "one shoot" process, knowledge refinement is a continuous process which is performed during the entire life-span of an expert system. In fact, the life-span of an expert system is directly determined by the validity of its knowledge base, which in turn is

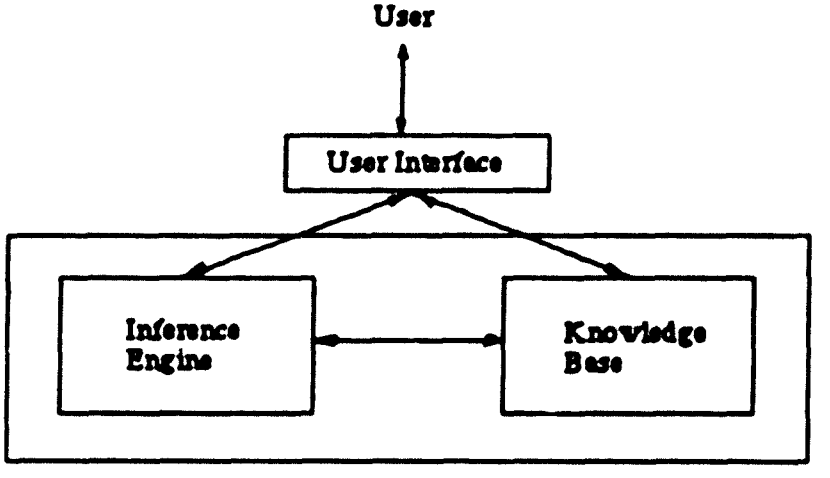

Fig. 2.1. Conventional Organization of an Expert System

determined by the extent knowledge is refined to match up with the changing domain. Knowledge refinement is a major issue in the study of expert system and is of great concern to both practitioners and academics. However, not much work have been done so far in this area.

2.2. Knowledge Refinement

The principle of knowledge acquisition as discussed in Chapter 1 is generalized to knowledge refinement. We define the process of knowledge refinement as a sequence of knowledge acquisition tasks with each triggered by an updating signal generated by the user or by the expert system itself. The idea of updating signals is to alert the expert system that the validity of the knowledge base has fallen below a threshold level and need to be revised.

2.2.1. A Measure of Knowledge Base Validity

The validity of knowledge base is difficult to measure on an absolute basis because expert system is basically a program that imitate the reasoning process of human experts. The best benchmark of the performance of an expert system is that of human experts. The validity of a knowledge base could then be mesaured by comparing the conclusions

generated by the expert system with those proposed by human experts in solving identical problems. For instance in medical diagnosis, diagnostic decisions of the expert system and the physicians can be compared to identify any discrepancy between the two parties. If the conclusions are so far off, then the knowledge base of the expert system is concluded to be invalid and has to be revised.

Using human experts as the oracles, we can provide a benchmark in assessing the performance of an expert system. Since conclusions are deduced from the knowledge base, the performance of an expert system could be determined by the validity of the stored knowledge. This permits us to use the performance measure to assess indirectly the validity of the knowledge base. However in using the same measure to evaluate performance and knowledge base validity, we have implicitly made the assumption that the inference mechanism of the inference engine does not deviate much from human experts. This assumption is justified in the sense that the way inference is made does not change as drastically as the knowledge itself in most situations. Yet we should bear in mind that there is still a chance that the declining performance of an expert system is not due to an invalid knowledge base but rather to an invalid inference mechanism.

2.2.2. Updating Signals

Updating signals are generated internally by the expert system or on request by users. Updating requests are generated internally when the performance of an expert system has fallen below a threshold level. To implement this, we need a log of conclusions generated by the expert system and audit them periodically. Depending on the degree of accuracy required, all elements in the log is used or only a portion of which is used in performance estimation. In the latter case, we need to determine the sample size and the acceptance level of the sample.

Expert systems organized in this way are self-controlled systems that adjust themselves to changing domains by responding to feedbacks from users or signals generated internally (Fig. 2.2.). The performance of an expert system is the control parameter of the entire system. It sets forth the criterion of revision and its value is determined by the log of previous conclusions.

The knowledge base can also be updated on request by users. This happens when 1) a user wants to change the threshold value of the performance measure, or 2) a user want to add or delete knowledge from the knowledge base. In both cases, the validity of the knowledge base may

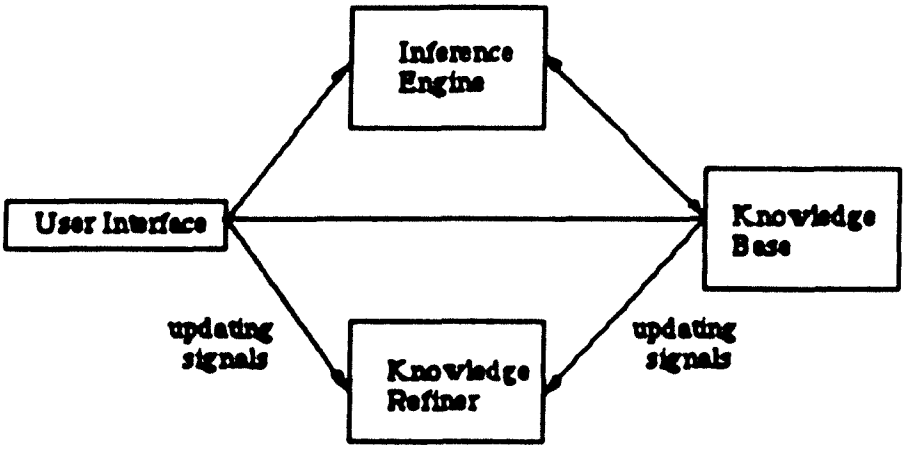

Fig. 2.2. Knowledge Refinement using Updating Signals

no longer in compliance with the given threshold value and need to be updated

2.3. An Architecture to Integrate Knowledge Refinement in Expert Systems

A generic architecture that support knowledge refinement is shown in Fig. 2.3. It consists of five functional components : 1) interface, 2) inference engine, 3) knowledge base, 4) knowledge refiner, and 5) decision log. The first three are the basic components of an expert system. The knowledge refiner and decision log are additional components that provide the mechanism and the information to refine the knowledge base.

The interface provides input/output media between users and the other components. It should provide user friendly interface that are easy to comprehend. It can be streamlined to certain group of end users and may take the form of icons, queries, natural language, graphics, image recognition, or real time signals from other devices.

The reasoning function is performed by the inference engine. It is implemented as a control procedure that determines how knowledge is combined and processed. The reasoning process can be forward or backward. In a forward reasoning process, knowledge is deduced from a

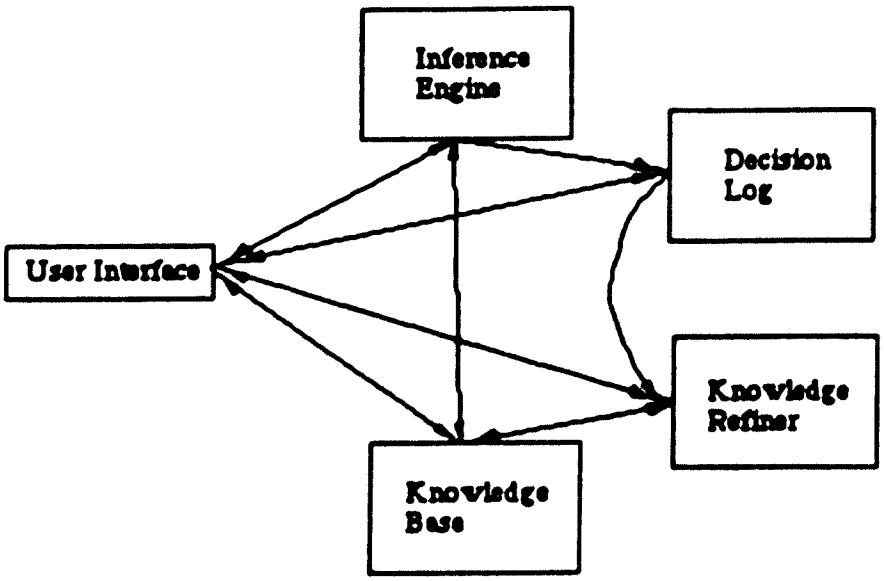

Fig. 2.3. An Expert System Architecture with Knowledge Refinement

collection of known facts. In backward reasoning, the process is reversed : it starts from a hypothesis and check whether it can be validated from the existing knowledge base. In a rule based system, they are commonly referred to as forward and backward chaining respectively.

In a knowledge base, there are many forms of knowledge including procedural knowledge, reasoning knowledge, presentation knowledge, and model knowledge. This is the reasoning knowledge that distincts a knowledge base from a conventional database system. Thus, the proposed architecture is primarily concerned with reasoning knowledge in the context of knowledge refinement.

The function of the decision log is to keep track of the changing reasoning process of human experts. Decisions generated by an expert system is stored in the log. By doing so, an expert system can evaluate its performance in terms of the number of contradictory decisions between human experts and the expert system. Decisions are recorded in the form of reasoning chain. A reasoning chain is the sequence of inference steps that lead to a decision. Entries in the decision log are audited periodically by experts to check for contradictory decisions. When a decision is audited, the entire reasoning chain of the decision is examined. Conflicting

decisions along the reasoning chain are identified and updating signals are then generated accordingly to refine the knowledge base.

Once the number of contradictory decisions in the decision log has fallen below a tolerance (threshold) level, an updating signal will be generated to inform the knowledge refiner to revise the knowledge base. The function of a knowledge refiner is to update the knowlege base using the knowledge acquisition techniques discussed in the previous chapter. The input of the knowledge refiner are the conflicting decisions and the existing knowledge base. The output would be a new knowledge base that satisfies the tolerance level.

Like the other components, the knowledge refiner and the decision log are permanent integrated components of an expert system and operate on a continuous basis.

2.4. Concluding Remarks

We have extended the conventional architecture of expert system to incorporate a mechanism of knowledge refinement. Two functional components , decision log and knowledge refiner, are added to provide the information and mechanism to refine the knowlege base respectively. In the next two chapters, task specific reasoning systems are studied under

this architectural framework. The two tasks are pattern recognition and classification. Furthermore, these two tasks are applied to the domain of security trading. Our intent is not to study security trading per sec but rather to provide a realistic setting to demonstrate the applicability of the proposed architectural framework in different task domains.

CHAPTER 3

PRICE MOVEMENT PATTERNS REPRESENTATION, ACQUISITION, AND REFINEMENT

3.1. Introduction

In this chapter, we will study the task of pattern recognition in the context of security trading. We will primarily focused on the following four issues : 1) Pattern knowledge representation, 2) Pattern recognition mechanism, 3) Noisy pattern discrimination, and 4) Pattern knowledge acquisition and refinement. Before we procede, let's briefly review the practice of price movement pattern recognition, commonly called technical analysis in the security industry.

3.1.1. Background

In formulating a trading strategy, a trader may study the movements of security prices in the hope of discovering trends and patterns on which he can capitalize. Numerous techniques have been proposed to aid traders in this aspect. They range from simple high/low point charts to complex trend-following programs that require

considerable computing resources. These techniques are collectively referred to as technical analysis. The practice of technical analysis is not uncommon, despite its lack of theoretical justification. The common denominator of these techniques is their assumption of the validity of historic information in predicting price movements. This is contrary to the Market Efficiency Hypothesis which contends that movements of security prices follow a random process. Yet, it does not obscure the fact that technical analysis is used daily by traders across a broad range of securities. Evidently and practically, it serves the function of decision support rather than "fortune teller" as most theorists perceive.

Automation of technical analysis is at the present limited to trend-following programs. These programs use statistical techniques such as moving average and time series to forecast future prices according to past trends, alerting the trader when the actual price movement deviates from the forecasted trend. Trend following is limited in the sense that it can only detect deviations from the trend. Except for some simple ones, it is not capable of identifying more complex trends. More complicated patterns (eg. valley, head and shoulder, Elliot wave etc.) still have to be "eyeballed" by traders. In general, technical analysis is primarily concerned with recognizing patterns of price movements. In addition to

the basic set of patterns (e.g. head and shoulder, valley) commonly used in technical analysis, a trader might add his own set of patterns.

Under our refined view of technical analysis, which is pattern recognition, the task of automating technical analysis is reduced to provide the followings :

1) Pattern representation form

2) Mechanism to recognize and discriminate different patterns

3) Mechanism to handle noisy patterns

4) Techniques to acquire and refine pattern knowledge

In this chapter, formal language theory is applied to address these four issues. The rest of the chapter is organized as follows : Section 3.2. reviews the application of statistical discriminant analysis in pattern recognition and discusses its drawbacks. In Section 3.3., formal language is used to describe price movement patterns, and the task of pattern recognition is reduced to parsing a sentence with a set of phrase structure grammars. Section 3.4. will address the issue of pattern knowledge acquisition and refinement from an architectural perspective based on Chapter 2.

3.2. Statistical Pattern Recognition

One approach to the task of pattern recognition is statistical discriminant analysis. A pattern such as the one in Fig. 3.1. is segmented into n-1 identical divisions. Thus, the continuous time span of a pattern is characterized by n discrete time spots. A pattern is represented by an n-dimensional vector $P = (h(t_1),h(t_2),...,h(t_n))^T$ where $h(t_i)$ is the price of the security at time t_i. The time interval between consecutive time spots is determined by the trader and it varies according to the time frame associated with a trading strategy. For instance, an arbitrage strategy in foreign exchange might require a very short time interval, say fractions of a second. Given a specific time frame, different patterns are represented by different pattern vectors. Price patterns are points in an n-dimensional Euclidean space called the pattern space. The task of pattern recognition is to partition the R^n pattern space. The objective is to group similar patterns into the same partition and to provide decision rules that classify patterns into different pattern classes. These decision rules take the form of discriminant functions. For each partition which represents all the possible forms of a pattern (e.g. valley), it is associated with a discriminant function D. D is a function

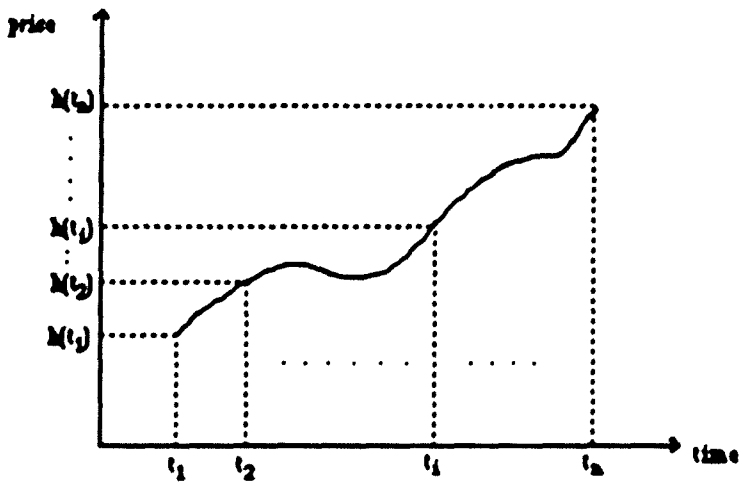

Fig. 3.1. Price Movement Segmentation

which maps R^n to R. Suppose the pattern space is divided into M partitions $G_1, G_2, ..G_M$, with each corresponding to a pattern class. There will be M discriminant functions $D_1, D_2, .. , D_M$. To recognize a price pattern, say P, the M discriminant functions are applied to V_p - the pattern vector representing P. The values of $D_1(V_p)$, $D_2(V_p)$,.. $D_M(V_p)$ are then compared with each other to determine to which partition V_p belongs. Suppose D_i, $1 \leq i \leq M$ represents the function that measures how likely V_p is an element of G_i ; the decision rule might be defined as follows :

V_p --> G_k such that $D_k(V_p) = \max \{D_1(V_p), D_2(V_p), ...D_M(V_p)\}$, $1 \leq k \leq M$

If k assumes more than one value, then one pattern class is arbitrarily selected and V_p is assigned to it. This approach to pattern recognition is also called the decision theoretic approach.

In general, the task is composed of two procedures, namely classifier construction and evaluation. Classifier construction is concerned with the construction of pattern boundaries which are based on a set of training samples. The partitions so constructed are evaluated by another

set of pattern samples to estimate how well the classifier performs in discriminating different patterns.

3.2.1. Classifier Construction

Given M classes of patterns, the entries of the training set are samples of each of the M patterns. For instance, a head and shoulder pattern might exist in slightly different forms as shown in Fig. 3.2. The function of this procedure is to consider a number of different head and shoulder patterns and to provide a compact form that adequately represents the entire class of head and shoulder patterns under a particular time frame. This is then repeated for other pattern classes. Obviously, the result depends very much on how fairly the sample represents the actual pattern. Bias samples will certainly degrade the accuracy of the procedure, and the result obtained will be misleading.

Statistical techniques come into play by assuming each pattern class is associated with a probability distribution. The most common approach is to assume that elements of a pattern class are distributed normally in R^n. Consider the case of discriminating M patterns.The first step of constructing a classifier is to estimate the parameters of the

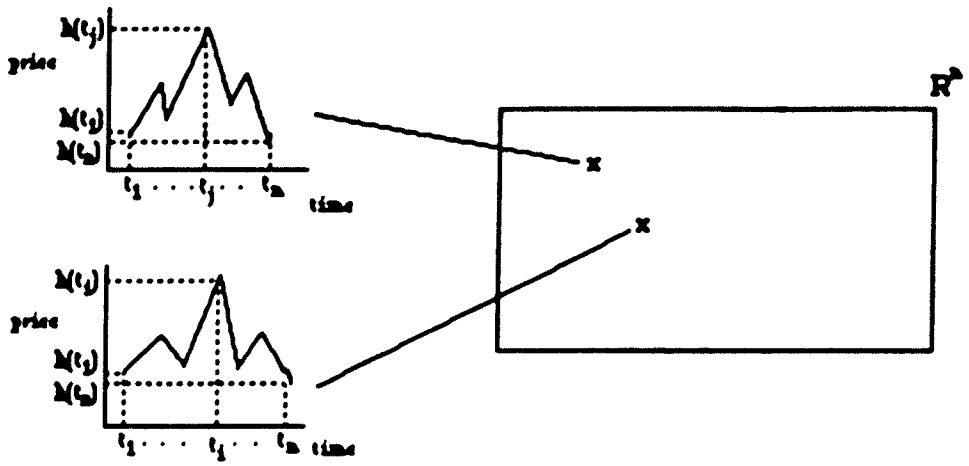

Fig. 3.2. Two Head and Shoulder Patterns in the Pattern Space

multivariate normal distributions, $N(\mu_1,\Sigma_1)$, $N(\mu_2,\Sigma_2)$,...$N(\mu_M,\Sigma_M)$. The sample patterns in the training set are used for the estimation.

Using the sample mean vectors $\mu_1',\mu_2',...\mu_M'$, and sample covariance matrices $\Sigma_1',\Sigma_2',..\Sigma_M'$, one obtains unbiased estimates of the parameters. The next step is to construct boundaries that separate these disributions. The construction of boundaries is driven by some predefined discriminant criteria. These criteria might be to minimize the maximum error, minimize total error, and to maximize interpartition distances etc.

Let us consider one that minimizes the expected cost of misclassification. The expected cost of misclassification, denoted by ECM, is defined as follows :

$$ECM = \sum_{i=1}^{M} p_i \left(\sum_{\substack{j=1, \\ j \neq i}}^{M} P(j \mid i) \right), \text{ in which} \qquad (3\text{-}1)$$

p_i = the prior probability of the i pattern class,

$P(j \mid i)$ = probability of misclassifying a pattern of the i class to the j class.

The objective is to determine partitions $G_1,G_2,..G_m$ in R^n in such a way that

(3-1) is minimized. It can been proved that minimizing (3-1) is defined by allocating $x \in R^n$ to G_k, k =1,..,M, for which

$$\sum_{\substack{i=1,\\ i\neq l}}^{M} p_i f_i(x), \tag{3-2}$$

$f_i(x)$ = probabality function of G_i

is smallest. (3-2) is smallest when the omitted term, $p_k f_k(x)$ is the largest. Thus, we have the following decision rule :

$$x \dashrightarrow G_k \text{ if } p_k f_k(x) > p_i f_i(x),\ i \neq k \tag{3-3}$$

Since $p_k f_k(x)$ and $p_i f_i(x)$ are always positive, (3-3) is equivalent to (3-4).

$$x \dashrightarrow G_k \text{ if } \ln(p_k f_k(x)) > \ln(p_i f_i(x)),\ i \neq k \ ,\ 1 \le i \le M \tag{3-4}$$

Here, $f_i(x)$ is a multivariate normal probability function. That is,

$$f_i(x) = (2\pi)^{-1/2} \mid \Sigma_i \mid \exp(-(x-\mu_i)^T \Sigma_i^{-1} (x-\mu_i)/2)\ ,\ 1 \le i \le M \tag{3-5}$$

We then have the following decision rule :

$$x \dashrightarrow G_k \text{ if } D_k(x) > D_i(k),\ i \neq k, \tag{3-6}$$

where $D_1, D_2, \ldots D_M$ are the discriminant functions obtained by substituting (3-5) in (3-4).

$$D_i(x) = -(\ln | \Sigma_i |)/2 - (x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)/2 + \ln p_i, \quad 1=1,2,...M \qquad (3\text{-}7)$$

Estimates of p_i, p_i', i=1,..M, can be obtained by counting the number of patterns belonging to G_i from the pooled pattern sample set. Unbiased estimates of Σ_i and μ_i, $1 \leq i \leq M$, can be obtained from sample mean vectors μ_i' and sample covariance matrices Σ_i', $1 \leq i \leq M$. Using these estimates, we obtain the estimated discriminant functions D_1', D_2', ... D_M' as follows :

$$D_i'(x) = -(\ln | \Sigma_i' |)/2 - (x-\mu_i')^T\Sigma_i'^{-1}(x-\mu_i')/2 + \ln p_i', \quad 1=1,2,...M \qquad (3\text{-}8)$$

The output of this optimization procedure consists of hyperplanes separating the different pattern classes in the pattern space. These hyperplanes are defined by the estimated discriminant functions which are linear.

3.2.2. Evaluating Classifier

The discriminant functions so derived have to be evaluated with another set of sample patterns. There are a number of techniques for testing the discriminating capability of these functions. One commonly used technique is to divide the training sample and testing sample into

equal halves. Performance of a classifier is calculated as a ratio of the number of misclassifications to the size of the testing sample. Quite obviously, the larger the size of the training sample, the less likely the discriminating functions will be distorted by bias sample patterns. Using the training sample for testing will probably underestimate classification error and should be avoided. Given a fixed number of sample patterns for each pattern class, tradeoffs have to be made between the size of the training sample and that of the testing sample.

A general procedure for constructing a pattern classifier using statistical techniques is depicted in Fig. 3.3. This approach, however, falls short in discriminating structural identical patterns under different time frames. Consider the two valleys shown in Fig. 3.4. They share the same structural form but only differ in their time frames. Note that our previous discussions on pattern classification assume that all patterns have the same time frame. One solution to this problem is to extend the time frame of the smaller pattern to that of the larger one. The process is called normalization of patterns. The general procedure is then applied to the normalized pattern samples. The classification error so obtained might be very large because the two pattern vectors might locate far from each other in the pattern space. A partition defined to contain these two vectors

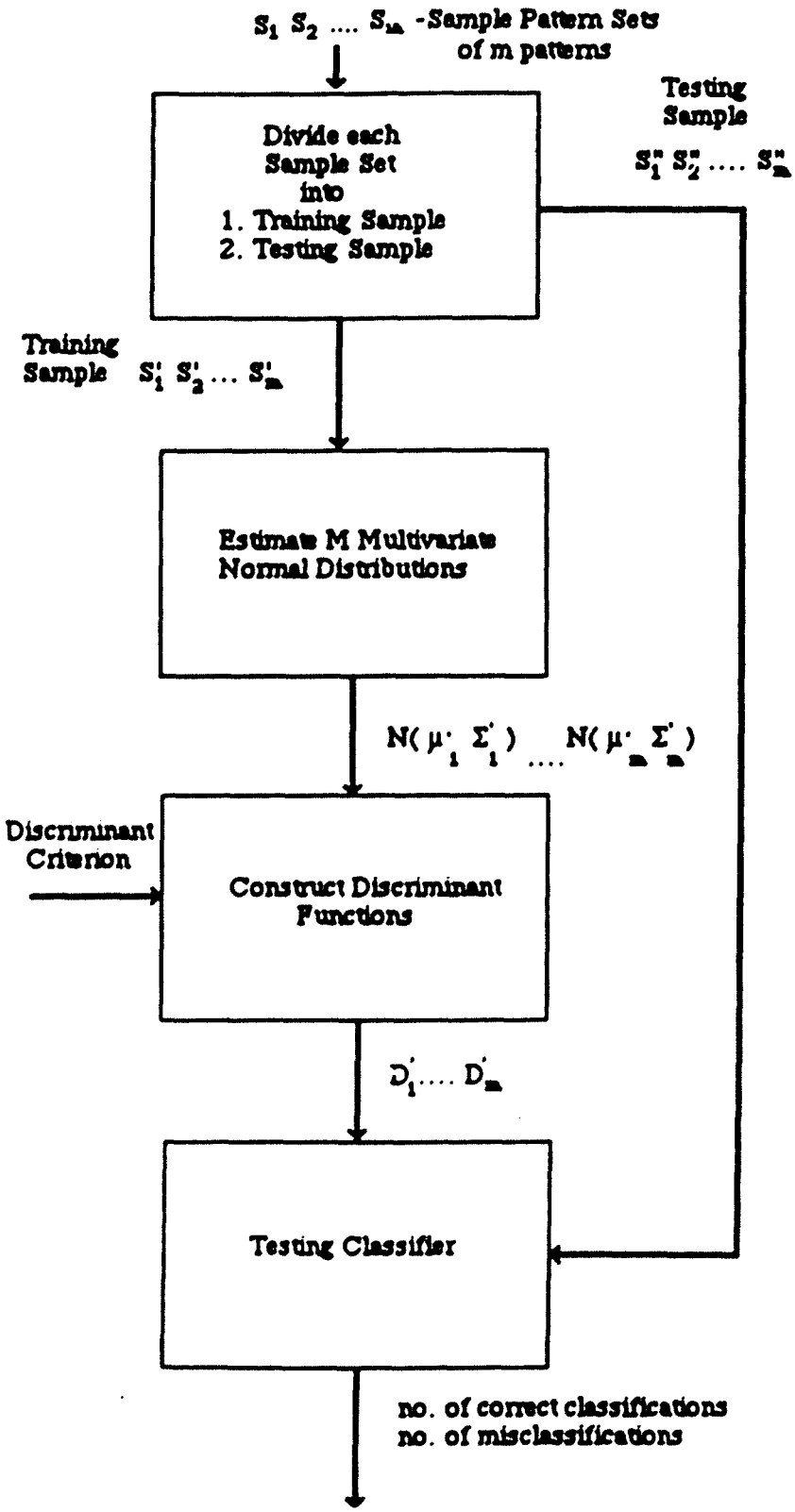

Fig. 3.3. Procedure for Constructing Pattern Classifiers Using Statistical Discriminant Analysis

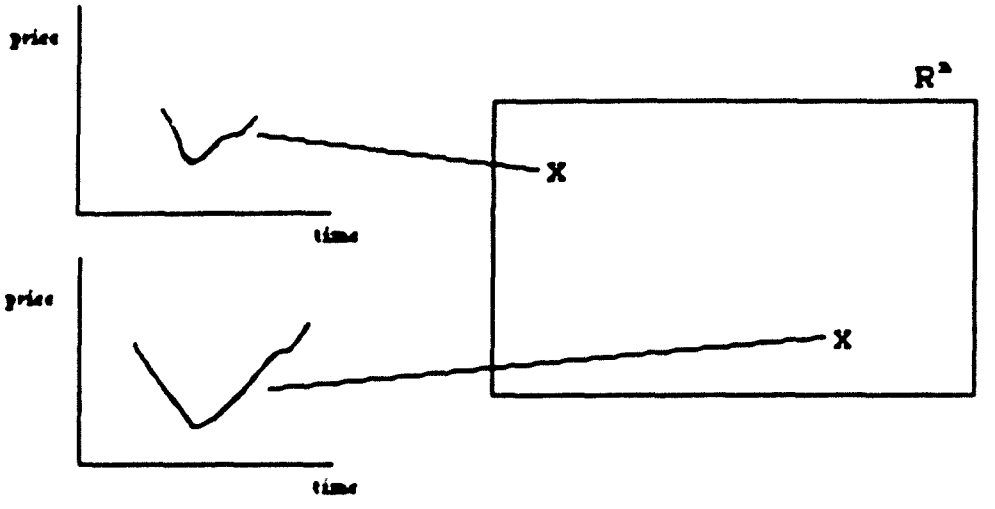

Fig. 3.4. Two Valleys with Different Time Frames

will probably be so large that a number of other patterns will also be included, thus increasing the error rate of the classifier. To overcome this, structural identical patterns under different time frames are characterized by partitions in different pattern spaces. In other words, pattern spaces of different dimensions are constructed for structurally identical patterns with different time frames. Consequently, the decision rule is stated as

"If a pattern vector falls within either of these partitions then conclude V_p belongs to this pattern."

This approach drastically increases the number of pattern partitions and becomes very inefficient when the number of time frames under consideration is large. To automate the task of pattern recognition effectively, we need a compact way to represent structurally identical patterns. In other words, we need constructs to state a pattern which is independent of the associated time frame. In the next section, this issue is addressed from the perspective of formal language theory.

3.3. Linguistic Pattern Recognition

The linguistic approach to pattern recognition in this section employs lingusitic techniques of formal language theory in stating and recognizing different patterns. Formal language theory is the study of the mathematical structure of sets of strings [Aho and Ullman 1968]. The initial investigation of mathematical lingusitics aimed at trying to understand the basic properties of natural languages [Chomsky 1956]. In general, a language is defined as the set of sentences that can be derived from a set of symbols and a collection of rewriting rules (or grammar rules). The grammar rules determine how symbols are combined to form sentences of the language. The number of sentences so generated may be finite or infinite. Thus, mathematical lingusitics is a powerful tool to describe a large number of phenomena or patterns by using a finite set of symbols and a relatively small number of grammar rules. In fact, it has been applied in a number of engineering domains such as fingerprint identification [Moayer and Fu 1976], image analysis [Gips 1974], [Rosenfeld 1979], [You and Fu 1979], and character recognition [Stallings 1979], to name a few.

The lingusitic approach presented here differs from its statistical counterpart in the way structural information of a pattern is used in the

recognition process. In this approach, structural information of a pattern is organized in the form of a hierarchy. In essense, a pattern is derived hierarchically from a predefined set of primitive patterns. The derivation process is governed by a collection of grammar rules.

To illustrate this, the class of valleys can be described by a language with its grammar rules shown below.

<valley> ---> <uptrend><downtrend>

<valley> ---> <downtrend><valley><uptrend>

<uptrend> --> /

<downtrend> ---> \

Two valleys with different sizes and their derivations are shown in Fig.3.5.

In this paper, formal language theory is applied to the study of technical analysis. Different price movement patterns are described by different languages. Each class of patterns , say valley, is associated with a pattern language. Sentences belonging to such a language represent all possible valley patterns which are independent of the underlying time frames. As will be shown later, the recognition process is reduced to

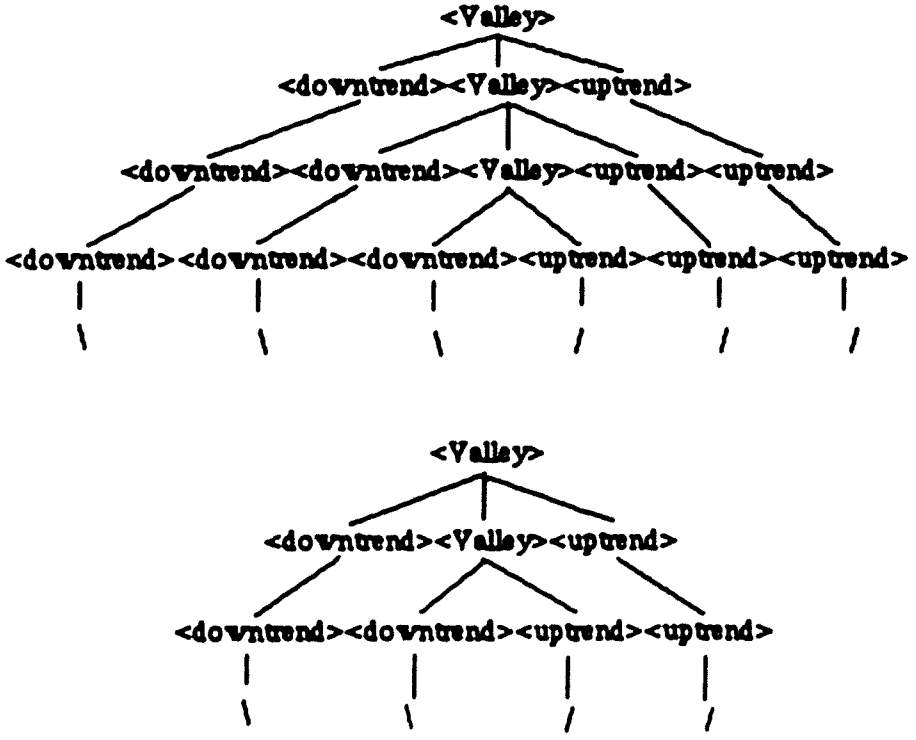

Fig. 3.5. Two Valley Sentences with Different Time Frames and Their Derivations Trees

parsing a pattern (coded as a sentence) with respect to the different pattern languages.

3.3.1. Representing Patterns as Languages

The langauge we use to describe a pattern belongs to the class of language that can be generated by a phrase structure grammar [Aho and Ullman 1968]. A phrase structure grammar is defined as a four tuple G, where

$$G = < V_T, V_N, P, S >, \text{ in which}$$

1. V_N and V_T are the nonterminal and terminal vocabularies of G, respectively.
2. $V_N \cup V_T$ constitutes the total vocabulary set V of G and $V_N \cap V_T = \phi$.
3. P is a finite set of grammar rules denoted by α --> β, where α and β are strings over V with α containing at least one symbol of V_N.
4. S is the start symbol of a sentence and $S \in V_N$.

The language generated by G is defined as $L(G) = \{x \mid x \in V_T^* \text{ such that } S \Rightarrow^* x\}$. L(G) is called the phrase structure language associated with G. V_T^* is the set of finite-length strings of symbols in V_T, including λ, the symbol with zero length. The length of a sentence ξ, denoted by $|\xi|$ is the

number of symbols in ξ. $S \Rightarrow^* x$ means that x is derived by systematically rewriting S using the grammar rules in P.

A class of patterns, say valley, is represented by a phrase structure language. Elements of the language correspond to the different valley patterns. One of the advantages of using this technique over statistical pattern recognition is that it can represent an infinite number of structurally identical patterns in a very compact form. To state a pattern class is to define the four attributes V_N, V_T, P, and S of G.

3.3.2. Primitive Price Patterns

Primitive price patterns are the most basic components of a pattern. They correspond to the terminal symbols in V_T. In our previous example of a Valley, { \ , / } are the primitive price patterns we use to describe the class of valley patterns. The meaning of each symbol is provided by the trader. For example, "\" might denote that the asking price of IBM stock is down by $0.1 in the last minute. Or it might mean the ¥/$ exchange rate is down by 0.5 in the last second. The meaning of each symbol in V_T depends on the trading activity engaged in, which in turn determines the descriptive

power of the language. The descriptive power of V_T is determined by the following:

1) Time interval of a symbol -- Since a sentence is composed of symbols concatenating in a sequence, it describes a series of events occuring in discrete time intervals. In order to approximate the continuous movement of prices, we have to decide on the time granularity of a primitive pattern. A better approximation can be obtained by reducing the time interval associated with a primitive pattern, making the pattern recognition mechanism more sensitive to short term price fluctuations. However, primitive patterns which span short time intervals may very likely incorporate undesired noises into a pattern, thus reducing the noise immunity of the recognition mechanism.

2) Number of primitive price patterns -- Given a specific time granularity, a richer set of primitive price patterns is more descriptive, providing a better approximation to the actual price movement. In Fig. 3.6b., a richer set of primitive patterns is used, giving a better approximation to the actual price movement than Fig. 3.6a. which uses a subset of the former.

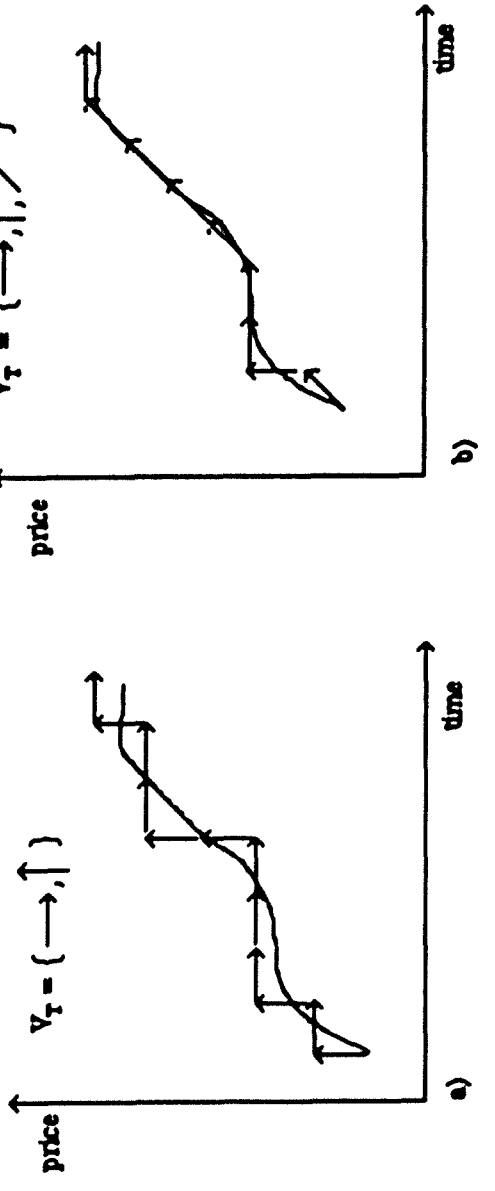

Fig. 3.6. Example of Primitive Price Patterns

3.3.3. Intermediate Price Patterns

Intermediate price patterns are patterns that are built up from primitive price patterns and other intermediate patterns. They represent the sub-patterns of a pattern, corresponding to V_N in G. For instance in our previous example, <downtrend> and <uptrend> are intermediate price patterns. They characterize the two sides of a valley. These patterns will not appear in the final pattern since all symbols in the final pattern are primitive patterns. They are used primarily as intermediate structures to construct the final patterns.

3.3.4. Grammar Rules Construction

The previous sections discussed the definition of V_T , V_N respectively.

How the various primitive and intermediate price patterns are combined to form more complex patterns are defined by a set of grammar rules, P. These grammar rules specify how patterns and sub-patterns are construct d.

According to Chomsky's hierarchy [1959a, 1959b] (Fig. 3.7.), phrase

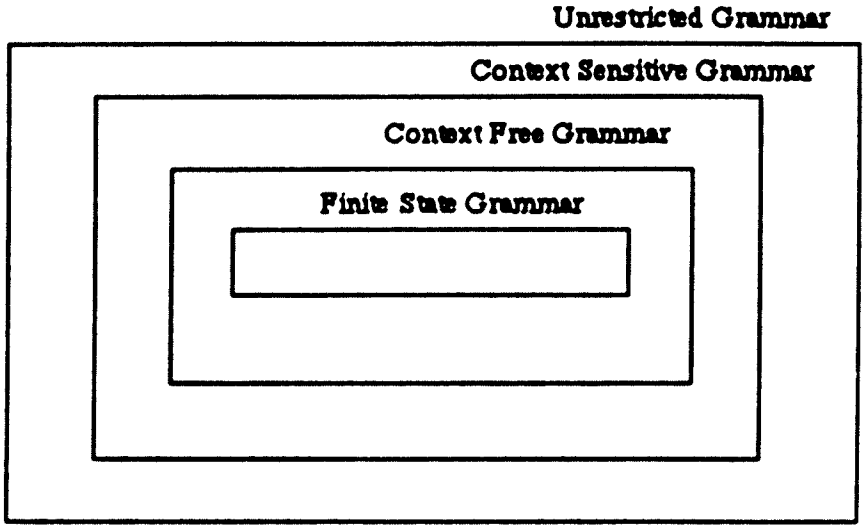

Fig.3.7. Chomsky's Hierarchy

structure grammar can be classified into four different classes according to the form of the grammar rules.

1) Unrestricted grammar - there is no restriction on the grammar rules, which might have any strings on either the right or the left of —>.

2) Context-Sensitive grammar - grammar rules are restricted to the form

$$\xi_1 A \xi_2 \Rightarrow \xi_1 B \xi_2$$

where $A \in V_N$, $\xi_1, \xi_2, B \in V^*$, $B \neq \lambda$, and $|\xi_1 A \xi_2| \leq |\xi_1 B \xi_2|$. That is, A can be replaced only in the context of ξ_1, ξ_2.

3) Context-Free grammar - grammar rules are of the form

$$A \text{ --> } B$$

where $A \in V_N$ and $B \in V^*$, and $B \neq \lambda$. Note that a grammar rule of such a form allows the nonterminal A to be replaced by the string B independently of the context in which A appears.

4) Finite-State grammar - The grammar rules are of the form

$$A \text{ —> } aB \quad \text{or} \quad A \text{ —> } b$$

where $A, B \in V_N$ and $a, b \in V_T$.

The question of which class of grammar to apply in describing a pattern class can be considered as one driven by the tradeoff between descriptive power and recognition efficiency. In terms of descriptive power, Unrestricted grammar is the most powerful amongst others. Since it properly contains other classes of grammars, it can describe patterns that cannot be described by other classes. However, determining whether a sentence is generated by an Unrestricted grammar is in general undecidable, not to say efficiency. On the other hand, sentences generated by a Finite-State grammar are easy to recognize. But its descriptive power is limited. Context-Sensitive and Context-Free grammars are intermediates which are recognizable and offer considerable descriptive power. In this thesis, we will focus solely on pattern classes that can be described by Context-Free grammars.

Using $V_T = \{ \backslash, / \}$, a number of pattern classes are defined below. As mentioned earlier, the semantics of "\" and "/" are defined by the trader. Notice that all these pattern classes are stated in Context-Free grammar.

Peak

By inverting a valley, one obtains a peak. A peak is defined as a pattern with an up trend followed by a down trend (Fig. 3.8a. and 3.8b.).

<Peak> --> <uptrend><downtrend>

<Peak> --> <uptrend><Peak><downtrend>

<downtrend> --> \

<uptrend> --> /

Zigzag

Using the pattern classes <Peak> and <Valley>, a pattern class called <Zigzag> which is defined as a sequence of peaks and valleys is shown below. Notice that Fig. 3.9a. and Fig. 3.9b. are not only different in their forms but also in their time frames. Classifier constructed using statistical discriminant analysis will very likely classify these two patterns into two different classes. Yet, they are generated by the same grammar, Zigzag.

<Zigzag> --> <Peak><Valley>

<Zigzag> --> <Valley><Peak>

<Zigzag> --> <Peak><Zigzag>

<Zigzag> --> <Valley><Zigzag>

<Valley> and <Peak> as described above.

Up-support

An up-support pattern is defined as a valley bounded by at least three "/" symbols on both sides. (Fig. 3.10a. and Fig. 3.10b.)

<Up-support> --> <up> <Valley><up>

<up> --> <uptrend><up>

<up> --> <uptrend><uptrend><uptrend>

<uptrend> and <Valley> as defined above.

Down-support

Similarily, a down-support pattern is defined below, consisting of a peak with both sides bounded by at least three "\" symbols. (Fig. 3.11a. and Fig. 3.11b.)

<Down-support> --> <down><Peak><down>

<down> --> <downtrend><down>

<down> --> <Downtrend><downtrend><downtrend>

<downtrend> and <Peak> as described above.

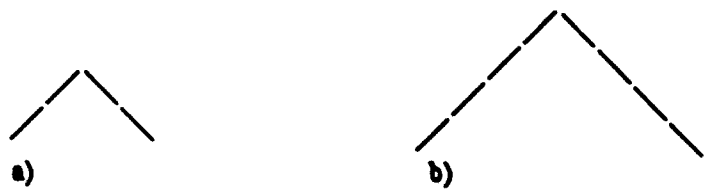

Fig. 3.8. A Peak

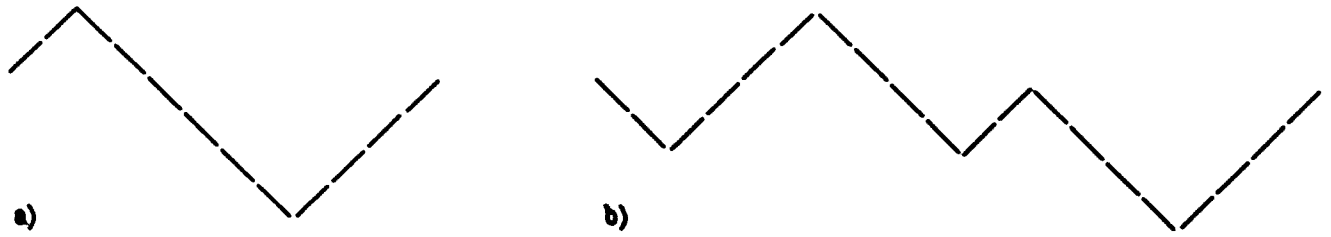

Fig. 3.9. A Zigzag

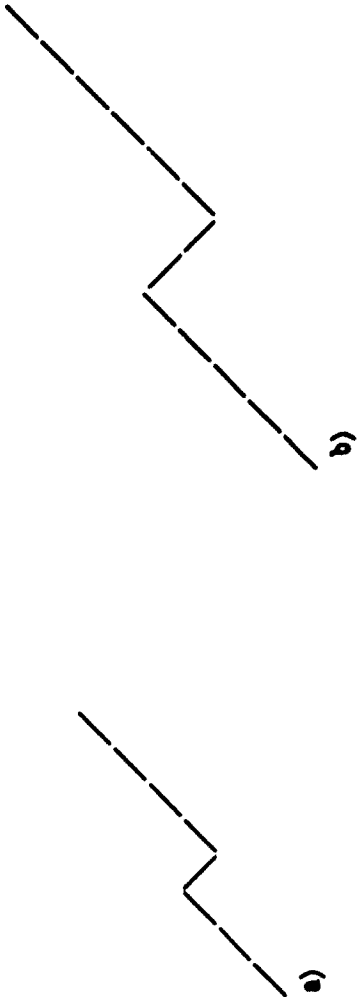

Fig. 3.10. An Up-support

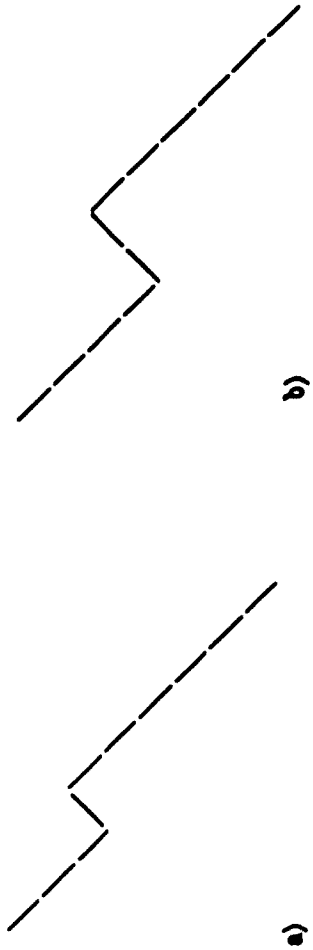

Fig. 3.11. A Down-support

3.3.5 Recognizing Patterns

We have discussed how structural identical patterns can be concisely represented by a phrase structure grammar. We now turn to the problem of recognizing price movement patterns which is stated as follows :

Input : (1) a price pattern coded as a sentence

(2) a collection of pattern classes $L(G_1)$, ... $L(G_M)$

Output : a decision showing which class the price pattern belongs to

In formal langauge, this is called the parsing problem. That is, given a sequence of symbols and a phrase structure grammar, determine whether the sentence can be derived by the grammar or not. Depending on the class a grammar belongs to, there are different parsing techniques called special purpose parsers. There are two generic approaches that these parsers are based on : Top-down parsing and Bottom-up parsing. In Top-down parsing, the symbol S is expanded by successively substituting nonterminals to try to fit the sentence. The goal is to discover a sequence of substitutions that transform S to the given sentence. In Bottom-up parsing, the method starts with the sentence and applies the grammar rules backward, trying to contract to the symbol S. In other words, the sentence is searched for subsentences that are the right parts of grammar rules. These are then replaced by the corresponding left side.

Here, we present a parsing technique called Earley Parser [Earley 1970] that is designed to recognize Context-Free language. Earley's algorithm is a Top-down parser that carries along all possible parses simultaneously in the form of parse lists.

Earley Parsing Algorithm

Input : A Context-Free grammar $G = <V_N, V_T, P, S>$ and an input string $w = a_1 a_2 a_3 ... a_n$.

Output : The parse lists $I_0, I_1, I_2, ... I_n$ for w.

Methods :

1. If S --> α is in P, add item [S --> .α,0] to I_0.

Perform 2 and 3 until no new items can be added to I_0.

2. if [B --> γ., 0] is in I_0, add [A --> αB.β,0] to I_0 for all [A --> α.Bβ,0].

3. if [A --> α.Bβ,0] is in I_0, add [B --> .γ,0] to I_0 for all B --> γ in P.

4. For each [B --> α.aβ,i] in I_{j-1} such that a = a_j, add [B --> αa.β,i] to I_j.

Perform steps 5 and 6 until no new items can be added.

5. if [A - > α., i] is in I_j, add [B --> αA.β,k] to I_j for all [B -->α.Aβ,k] in I_i.

6. if [A --> α.Bβ,i] is in I_j, add [B --> .γ,j] to I_j for all B --> γ in P.

where $\alpha, \beta, \gamma \in V^*$, A, B$\in V_N$.

Decision rule : if there are some items of the form [S --> α.,0] in I_n, then w is in L(G).

Fig. 3.12. depicts a system that can discriminate different patterns. First, a grammar is constructed for each language. Second, movements of prices are coded as a sequence of symbols (sentences). By parsing a sentence with respect to each grammar, it is possible to decide to which grammar (pattern class) the incoming sentence (price movement) belongs. Notice that there might be two undecisive outcomes of the procedure :

1) More than one grammar exists that can successfully parse the sentence, so a pattern is classified into more than one pattern class.

2) The sentence cannot be parsed by any grammar. In other words, the pattern is not classified to any of the given pattern classes.

The first situation occurs because $L(G_1),L(G_2),...L(G_n)$ are not disjointed. There are some sentences that are covered by more than one language, implying that these sentences can be generated by more than one grammar. There is no trival way to decide which class the pattern belongs to. A cla s can be chosen randomly or human judgments are resorted to in this situation. To avoid this undecisive result, care must be exercised in designing grammars in such a way that the final pattern classes are

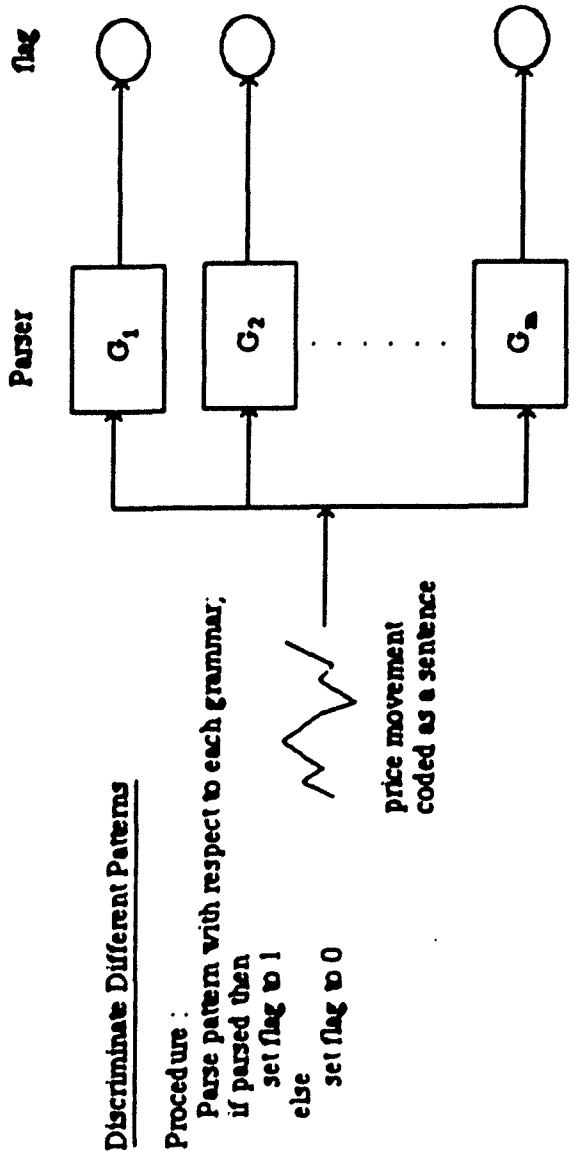

Fig. 3.12. Pattern Classifier

disjointed. Note that the grammar rules stated in the previous section are all disjointed. In the second case, the pattern is called a noisy pattern with respect to the given grammars. The pattern is distorted by the noise in such a way that it does not belong to any of the pattern classes. Yet, it is possible to decide which class a noisy pattern is most "likely" to be in. In the next sections, the problem of classifying noisy patterns is addressed and a procedure based on error-correcting parsing is used to discriminate noisy patterns.

3.3.6. Recognizing Noisy Patterns

The problem of classifying noisy patterns is to find a pattern class that the noisy pattern is most likely to belong to. A measure of likelihood or similarity between patterns is required to make comparisons. The measure is then extended to that between a pattern and a pattern class. The idea is to search for a pattern (sentence) from each pattern class (grammar) which is the most similar to the given noisy pattern. Amongst these patterns, the most similar pattern is chosen. The noisy pattern is classified to the class associated with the chosen pattern. This is depicted pictorially in Fig. 3.13.

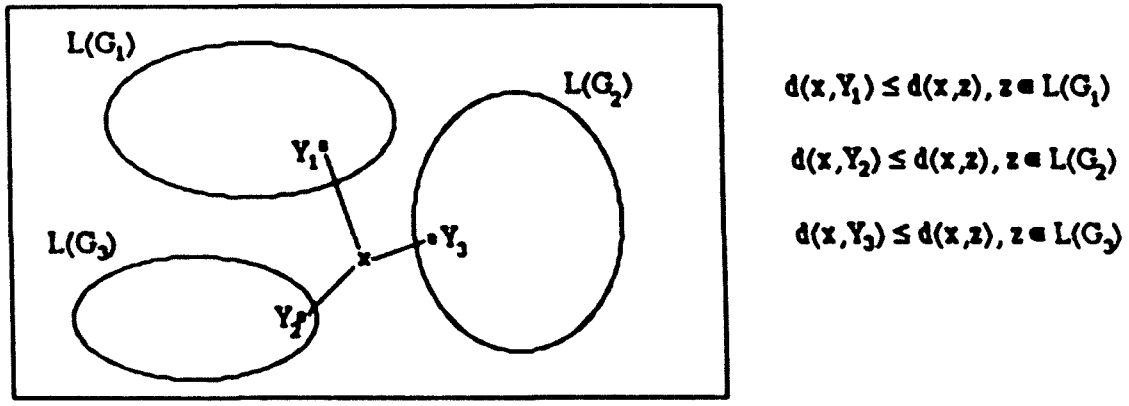

Fig. 3.13. Measuring the Distance Between a Sentence and a Language

3.3.6.1. A Measure of Similarity of Patterns

To measure the similarity of two patterns is equivalent to measuring how similar two sentences are. A metric to measure distance between sentences is adopted from [Levenshtein 1966]. The metric is defined on transformations. Three transformations are defined below :

1) Substitution transformation

$$w_1 \alpha w_2 \rightarrow^{s} w_1 \beta w_2, \qquad \text{for all } \alpha, \beta \in V_T, \alpha \neq \beta$$

2) Deletion transformation

$$w_1 \alpha W_2 \rightarrow^{d} w_1 w_2, \qquad \text{for all } \alpha \in V_T$$

3) Insertion transformation

$$w_1 w_2 \rightarrow^{i} w_1 \alpha w_2, \qquad \text{for all } \alpha \in V_T$$

where $w_1, w_2 \in V_T^*$.

The distance between two sentences, $x, y \in V_T^*$, denoted by $d_L(x,y)$, is defined as the smallest number of transformations required to derive y from x. The distance $d_L(x,y)$ is called the Levenshtein distance between two sentences. A weighted Levenshtein distance can be defined by assigning nonnegative weights w_s, w_d, w_i to substitution, deletion, and

insertion transformations, respectively. Let T be a sequence of transformations to derive y from x ; the weighted distance between x and y denoted by $d_{wL}(x,y)$ is defined as

$$d_{wL}(x,y) = \min_{T} \{ w_s n_s + w_d n_d + w_i n_i \}$$

where n_s, n_d, and n_i are the number of substitutions, deletion, and insertion transformations, respectively, in J. In cases when $w_s=w_d=w_i=1$, we have $d_{wL}(x,y) = d_L(x,y)$. For example, to transform \\\\\/\\\/ to a valley requires three substitutions (see Fig. 3.14a.). However, only one substitution is required to transform \\\\\/\\\/ to a down-support (see Fig. 3.14b.)

3.3.6.2 Minimum-Distance Error-Correcting Parsers

Given a grammar G and a sentence y, the essene of error-correcting parsing is to search for a sentence x in L(G) such that

$$d(x,y) = \min_{z} \{ d(z,y) \mid z \in L(G) \}$$

In other words, a minimum-distance error-correcting parser will find a price pattern associated with a pattern class which is the closest to the given pattern [Aho and Peterson 1972]. The procedure starts by expanding G to incorporate the three types of transformations in the form of

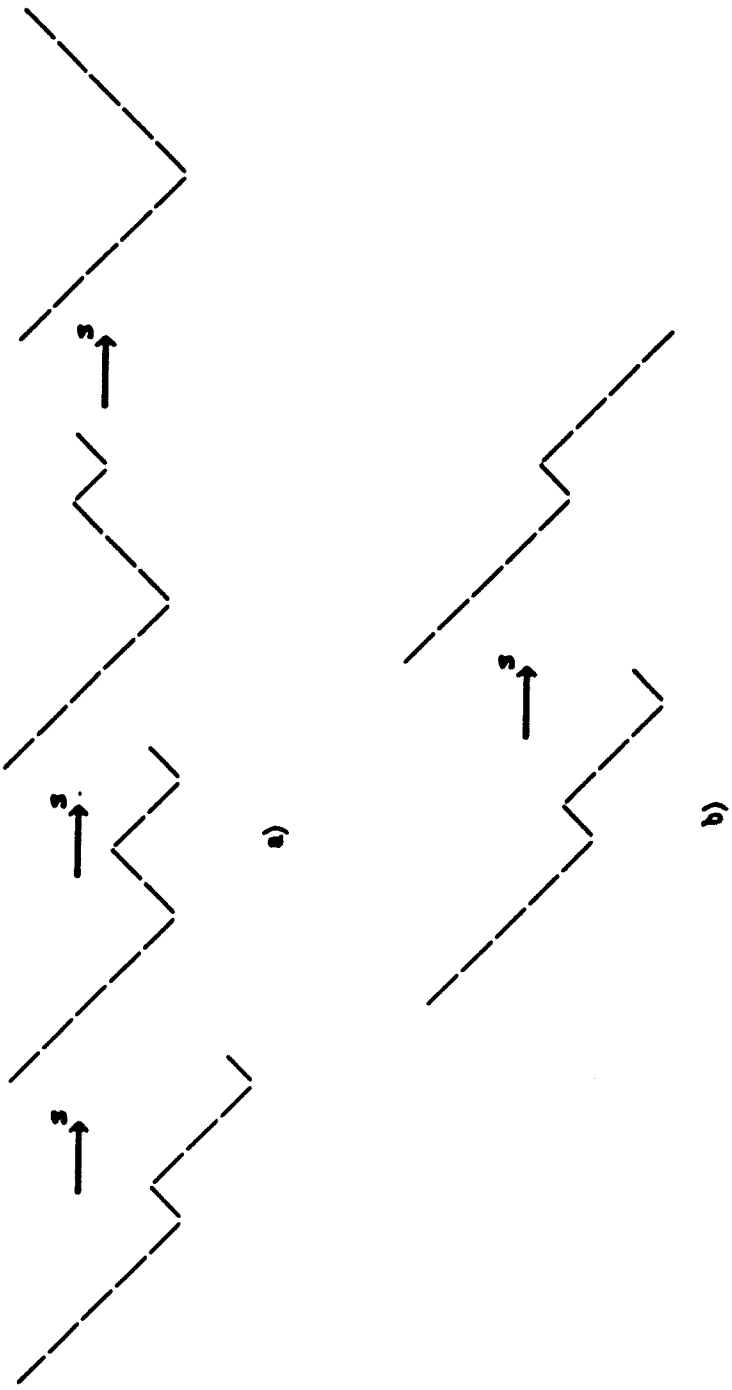

Fig. 3.14. Transformation of \\\\/\\\/ to a Valley and a Down-support

grammar rules, called error grammar rules [Tanaka and Fu 1978]. The expanded grammar G' includes not only L(G), but all sentences with the three types of transformations. The parser constructed according to G' with a provision to count the number of transformations used in a derivation is called the error-correcting parser of G. For a given y, the error-correcting parser of G will generate a parse which consists of the smallest number of transformations (or the smallest accumulated weight). A sentence x in L(G) that satisfies the minimum-distance criterion can be generated from the parse by eliminating the error transformations.

Construction of Expanded grammar G' from G

Input : a Context-Free grammar $G = <V_N, V_T, P, S>$

Output : $<V_N', V_T', P', S'>$, where P' is a set of weighted grammar rules

Methods :

1. $V_N' = V_N \cup \{S'\} \cup \{E_a \mid a \in V_T\}$, $V_T' \supseteq V_T$.

2. if $A \rightarrow a_0b_1a_1b_2....b_ma_m$, $m \geq 0$ is a grammar rule in P such that $a_i \in V_N^*$ and $b_i \in$ $_T$; then add $A \rightarrow a_0E_{b1}a_1E_{b2}....E_{bm}a_m$, 0 to P', where each E_{bi} is a new nonterminal, $E_{bi} \in V_N'$, and 0 is

the weight associated with this grammar rule.

3. Add the following grammar rules to P'.

Grammar Rule	weighted Levenshtein weight	
(1) $S' \rightarrow S$	0	
(2) $S' \rightarrow Sa$	δ	for all $a \in V_T$
(3) $E_a \rightarrow a$	0	for all $a \in V_T$
(4) $E_a \rightarrow b$	σ	for all $a \in V_T$, $b \in V_T$, and $a \neq b$
(5) $E_a \rightarrow \lambda$	γ	for all $a \in V_T$
(6) $E_a \rightarrow b E_a$	δ	for all $a \in V_T$, $b \in V_T$

The grammar rules in (2), (4), (5), (6) are called error grammar rules. Each error grammar rule corresponds to one type of error transformation on a particular symbol in V_T. Thus, it is possible to measure the distance between a sentence and a langauge by counting the number of error gramma ; (or accumulating the weights of grammar rules) used in the derivation.

For example, the expanded grammar of the valley pattern is shown below :

$G_V' = < V_T', V_N', P' S' >$, in which

V_T' = { \, -, / },

V_T' = { <Valley>', <Valley>, <uptrend>, <downtrend>, $E_/$, $E_\backslash$ },

S' = <Valley>'

P' :

Grammar rule	Levenshtein distance (ie. $\sigma=\gamma=\delta=1$)
<Valley> --> <downtrend><uptrend>	0
<Valley> --> <downtrend><Valley><uptrend>	0
<Valley>' --> <Valley>	0
<Valley>' --> <Valley>/	1
<Valley>' --> <Valley>\	1
<Valley>' --> <Valley>-	1
<uptrend> --> $E_/$	0
<downtrend> --> $E_\backslash$	0
$E_/$ --> /	0
$E_/$ --> -	1
$E_/$ --> \	1

$E_/$ --> /$E_/$	1
$E_/$ --> \$E_/$	1
$E_/$ --> -$E_/$	1
$E_/$ --> λ	1
$E_\backslash$ --> \	0
$E_\backslash$ --> -	1
$E_\backslash$ --> /	1
$E_\backslash$ --> \$E_\backslash$	1
$E_\backslash$ --> /$E_\backslash$	1
$E_\backslash$ --> -$E_\backslash$	1
$E_\backslash$ --> λ	1

A modified Earlier Parser [Tanaka and Fu 1978] for the expanded grammar G' with a provision to accumulate the weights associated with the grammar rules used in a derivation is presented as follows :

Minimum-distance error-correcting Earley Parser

Input : An expanded grammar $G' = (V_N', V_T', P', S')$ and an input string $y = b_1 b_2 b_3 \ldots_m$ in $V_T'^*$.

Output : $I_0, I_1, I_2, \ldots I_m$, the parse list for y, and d(x,y), where x is the minimum-distance correction of y.

Method :

1. Set j = 0. Then add [E --> .S',0,0] to I_j.

2. If [A --> α.Bβ, i, ξ] is in I_j, and B --> γ, η is a production rule in P', then add item [B --> .γ, j, 0] to I_j.

3. If [A --> α., i, ξ] is in I_j, and [B --> β.Aγ, k, ξ] is in I_i, and if no item of the form [B --> βA.γ, k, ϕ] can be found in I_j, then add an item [B--> βA.γ, k, $\eta+\xi+\zeta$] to I_j, where ζ is the weight associated with grammar rule A --> α. If [B-->βA.γ, k, ϕ] is already in I_j, then replace ϕ by $\eta+\xi+\zeta$ if $\phi >$ $\eta+\xi+\zeta$.

4. If j = m go to step 6; otherwise , j = j+1.

5. For each item in I_{j-1} of the form [A --> α.bjβ, i, ξ] , add item [A --> αbj.β, i, ξ] to I_j and go to step 2.

6. If item [E --> S'., 0, ξ] is in I_m, then d(x,y) = ξ, where x is the minimum distance correction of y. Exit.

where $\alpha,\beta,\gamma \in V'^*$, A, B $\in V_N'$

The minimum distance correction distance of y, which is x, can be derived from the parse by eliminating all the error grammar rules. A parse of \\\\ /\- / with the expanded valley grammar is shown in Fig. 3.15.

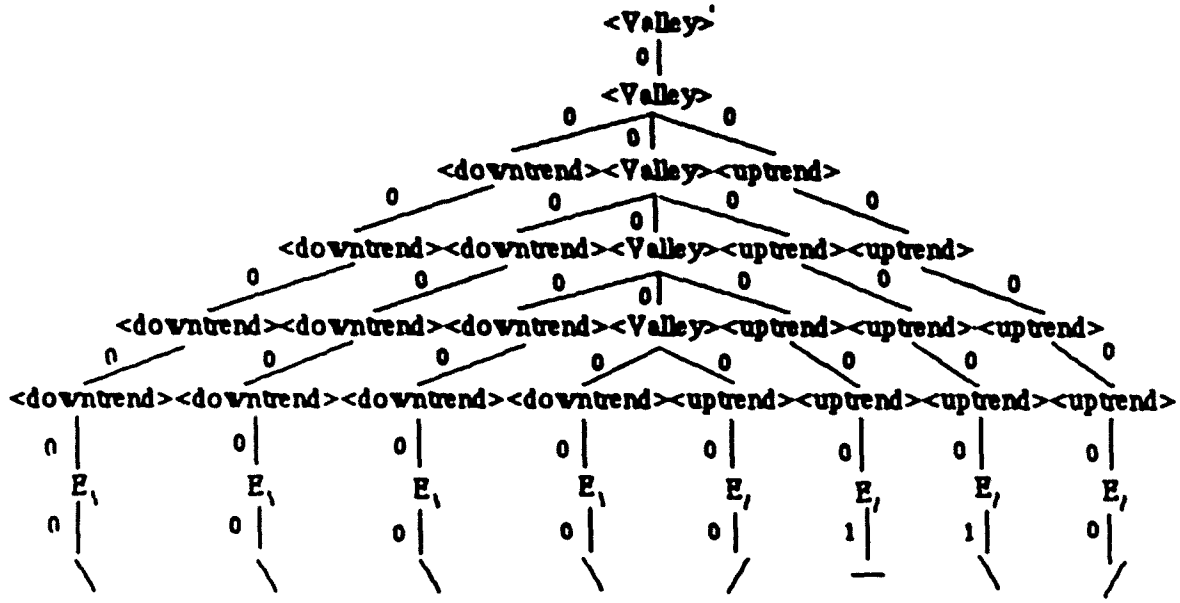

Fig. 3.15. A Parse Tree of \\\\V-V with the Expanded Valley Grammar

3.3.6.3. Classifying Noisy Patterns

By creating an expanded grammar for each pattern grammar and constructing an error-correcting parser for each, it is possible to construct classifiers that can discriminate noisy patterns. As shown in Fig. 3.16., a pattern, say y, is first parsed with respect to each error-correcting parsers G'_1, G'_2,...G'_M. The minimum error-correcting distance, $d(L(G_i),y)$, between y and each pattern class $L(G_i)$ is obtained and compared according to the following criterion :

y is classified to G_k iff $d(L(G_k),y) = \min \{ d(L(G_i),y) \mid i = 1...n\}$, $k \in \{1..n\}$

That is to say, y is classified to the pattern grammar that generates a sentence which is the closest to y among others.

3.4. Language Acquisition based on Grammatical Inference

So far we have been assuming that the pattern grammars are provided by the trainers. Depending on the pattern complexity, the task of specifying a grammar to describe a pattern can be very tedious and erroneous. Instead of acquiring these grammar rules manually, one can resort to

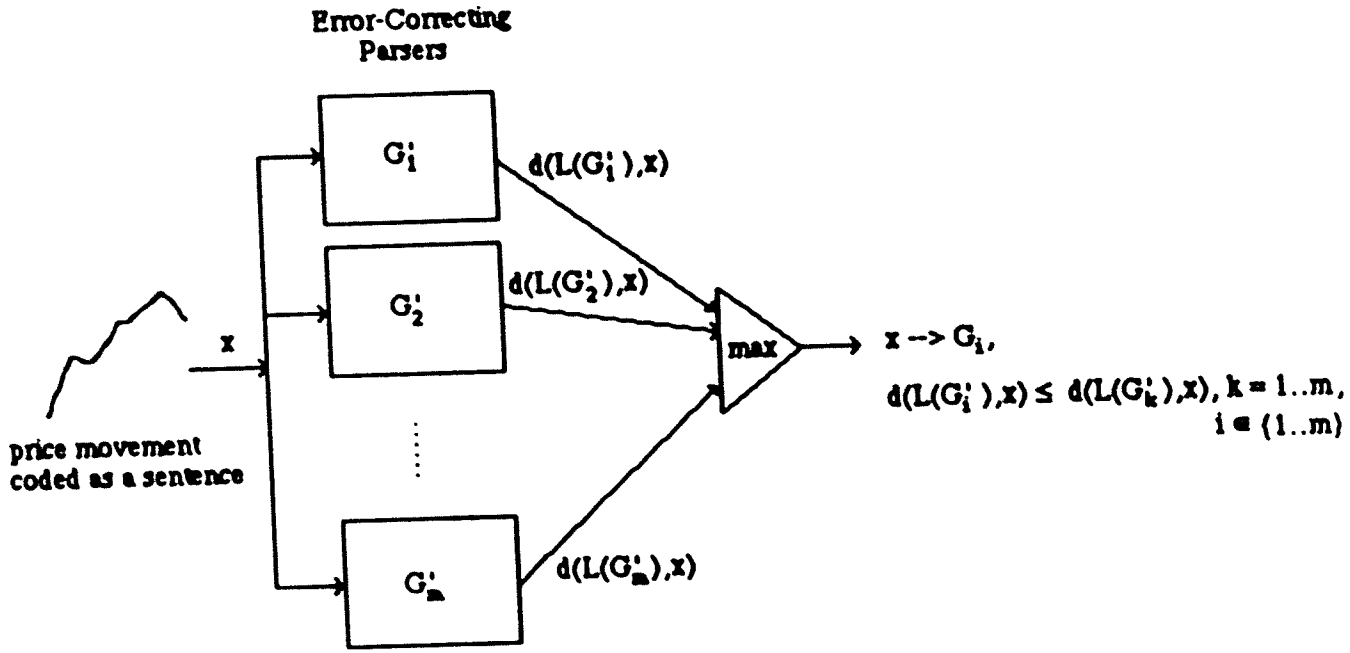

Fig. 3.16. Classifer for Noisy Patterns

automatic grammar rules inference techniques. The principle of these techniques is to derive the grammar of a lauguage by directly inferring from a set of sample sentences (patterns) of the language.

Grammatical inference is concerned with the procedure to infer the grammar rules G based on a finite set of sentences S_t from L(G), the langauge generated by G, and possibly also on a finite set of sentences from the complement of L(G). The schematic diagram of a grammatical inference procedure is shown in Fig. 3.17.

3.4.1. Grammatical Inference Procedure

The input to the inference procedure is a sample of a langauge L, denoted by $S_t(L)$. $S_t(L)$ is defined to be the set $\{+x_1,....+x_n\} \cup \{-x_1,....,-x_m\}$, where $S^+ = \{+x_1,....+x_n\}$ is defined to be the positive sample of $S_t(L)$, and $S^- = \{-x_1,....,-x_m\}$ the negative sample of $S_t(L)$. S^+ is said to be structural complete if every grammar rule in G is used to generate at least one sentence in S^+. The property of structural completeness is important because there is no one-to-one relationship between a grammar G and the langauge L(G) that generated by G. The assumption of having a

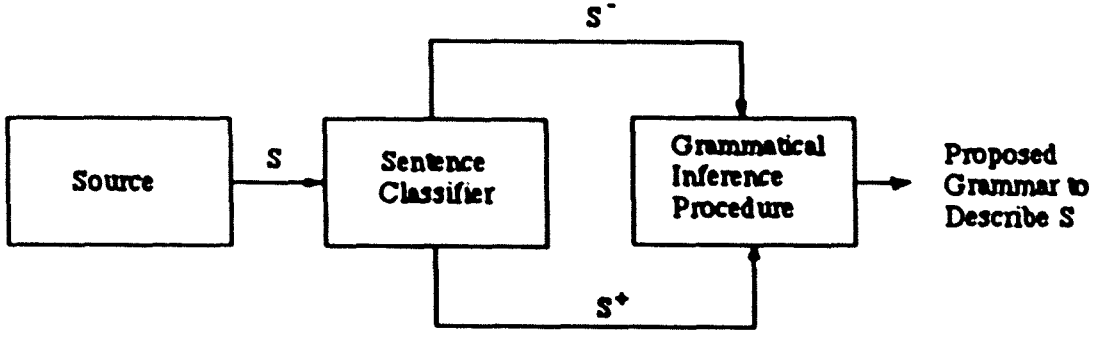

Fig. 3.17. Schematic Diagram of Grammatical Inference

structural complete S^+ will significantly reduce the size of possible grammars for S^+.

Furthermore, a class of grammar $\mathcal{G}$ is admissible if : 1) $\mathcal{G}$ is denumerable, i.e. $\mathcal{G} = \{G_1, G_2, ...\}$; and 2) for any $x \in V_T^*$, it is decidable whether or not $x \in L(G)$, for any G in $\mathcal{G}$. Admissibility is a criterion that determines the usefulness of the inferred grammars and needed to be satisfied. In general, S^+ is used to generate $\mathcal{G}$ while S^- is used to test the validity of each G in $\mathcal{G}$ in light of the given assumptions on grammar class.

The complexity of grammatical inference is dependent on the class of grammar to be inferred. Unfortunately, most of the inference procedures for the class of Context-Free langauge or above are heuristic in nature. Finite-State language is the most easy to deal with because most questions pertaining to Finite-State languages are decidable. However, some of these questions are proved to be undecidable in other langauge classes. For the rest of the chapter, we will primarily focus on Finite-State and Context-Free Grammar only.

3.4.1.1. Inference Procedure for Finite-State Pattern Grammar

The following procedure discusses the acquisition of Finite-State pattern grammar rules. The following assumptions are made :

1) The grammar being inferred is Finite-State

2) The sample of the language $S_t=(S^+,S^-)$ is a finite sample

3) The set S^+ is structural complete

4) The inferred grammar G is such that S^+ is a subset of G and S^- is a subset of the complement of G.

Using these assumptions, an admissible class of Finite-State grammars will be derived using S^+ and the following inference procedure. Different techniques will be used to select the inferred grammar from this admissible class as the inferred grammar. When $S^- \neq \phi$, the null set, the information in S^- can be used in the selection process.

The inference procedure discussed below will generate a Canonical Definite Grammar $G_c = <V_{CN},V_{CT},P_C,S_C>$ that can exactly describe a given S^+. The inference procedure is stated as follow :

1) Examine each $x \in S^+$ and identify all of the distinct terminal symbols used in the generation of the strings of S^+.

2) For each $x_i = a_{i1}a_{i2}....a_{in}$, $x_i \in S^+$, define the distinct set of rewriting rules

$S \dashrightarrow a_{i1}Z_{i1}$

$Z_{i1} \dashrightarrow a_{i2}$

.

.

$Z_{in-1} \dashrightarrow a_{in}$

Each Z_{ij} represents a new nonterminal symbol.

3) The set V_{CN} consists of S and all the distinct nonterminal symbols produced by step 2. The set of P consists of all the distinct rewriting rules defined by step 2.

The grammar $G = <V_{CN},V_{CT},P_C,S_C>$ defined by the above procedure is a Finite-State grammar which has the property

$L(G) = S^+$ and S^- is the subset of the complement of L(G)

For example, given $S^+ = \{\backslash\!/, \backslash/\}$ and $S^- = \phi$, the Canonical Definite Grammar G_C corresponding to S^+ is as follow :

$V_{CT} = \{\backslash/\}$, $V_{CN} = \{ S_C, Z_1, Z_2, Z_3, Z_4, Z_5\}$

P_C : S_C --> \$Z_1$ S_C --> \$Z_5$

Z_1 --> \Z_2 Z_2 --> /Z_3 Z_3 --> /Z_4 Z_4 --> /

Z_5 --> /

The Canonical Definite Grammar so constructed may result in a large nonterminal set. Some of the nonterminals in V_{CN} are redundant. For instance Z_4 and Z_5 in the above example can be combined and replaced by another symbol. By combining equivalent symbols, we can derive a more compact grammar for S^+.

It is possible to derive grammars from Canonical Definite Grammar by partitioning the nonterminal set V_{CN} into different equivalent classes. The grammars so constructed are called Derived Grammars. A Derived Grammar $G_D = <V_{DN}, V_{DT}, P_D, S_D>$ is defined as follows :

$V_{DN} = \{B_1, B_2, B_3, ...B_n\}$, $B_1 \cup B_2 \cup ... \cup B_n = V_{CN}$ and for any $i \neq j$

$B_i \cap B_j = \phi$,

$V_{DT} = V_{CT}$,

P_D is defined as :

1) A production of the form B_i --> aB_j is contained in P_D if and only if there exists Z_a, $Z_b \in V_{CN}$ such that Z_a --> a Z_b, $Z_a \in B_i$ and $Z_b \in B_j$.

2) A production of the form B_i --> a is contained in P_D if and only if there exists $Z_a \in V_{CN}$ such that Z_a --> a, $Z_a \in B_i$.

To illustrate , let G_D be the Derived grammar of the above Canonical Definite Grammar with $V_{DN} = \{B_1, B_2, B_3, B_4\}$ where $B_1 = \{ S_C \}$, $B_2=\{Z_1\}$, $B_3 = \{Z_2\}$, $B_4 = \{Z_3, Z_4, Z_5\}$, and the set P_D is defined as

B_1--> \$B_2$ B_1 --> \$B_4$ B_2 --> \$B_3$ B_3 --> /B_4 B_4 --> /.

There are more than one way to partition the nonterminals in V_{CN}, implying that there are more than one grammars that can be derived from a Canonical Definite Grammar. Since the number of derived grammars is finite and it is decidable to determine whether two Finite-State Grammars are equivalent, it is possible to enumerate all derived grammars and select the most compact one.

3.4.1.2. Inference Procedure for Context-Free Pattern Grammar

Context-Free grammar is more difficult to deal with because some of the questions that are decidable in Finite-State grammar are undecidable in Context-Free Grammar. For instance it is in general undecidable to determine whether two Context-Free grammars are equivalent or not. Most of the inference procedures for Context-Free gramamr are heuristic in nature. The following procedure as proposed by Solomonoff [1964] is one that infers the grammar of a sample of sentences using a teacher as the oracle.

This semi-automatic procedure will discover recursive grammar rules for a subset of Context-Free grammar. The procedure described by Solomonoff consists of two steps :

1) Delete substrings of a valid string and ask the teacher if the remaining string is acceptable. By acceptable, we mean the string is a proper sentence of the inferred langauge.

2) If the remaining string is acceptable, we reinsert the deleted substring with several repetitions and ask if the resulting string is acceptable. If the resulting string is acceptable, a recursive construction is formed.

Let's consider the following example: Let \V/ be a sample sentence, the teacher has to evaluate the validity of the following strings :

{V/ // / \ V \\ \V }

Suppose the first and the last characters are deleted from \V/, the remaining string (i.e. V) is evaluated to be valid. Then the teacher is queried as to the validity of strings \V/, \\V//, \\\V///, ..., \\..V../. If they are all acceptable for sufficiently large number of repetitions, the grammar rules {A --> \A/ A --> V} are inferred. Note that the Valley grammar is rediscovered by this procedure. However this procedure, like all heuristic inference procedures, can only infer Context-Free grammar of specific structure. Nevertheless, the inference procedure as described above can aid a trader to specify a pattern grammar if the grammatical structure is known beforehand.

3.4.2. Language Refinement using Grammatical Inference

The inferred grammar using the foregoing inference procedures probably will not the be exact grammar that identify the language if there exists one. It is due to the fact that the sample (S^+ S^-) may be bias and violate the assumption of structural completness. Although there is no way to guarantee that a sample is structural complete, we can approach the

exact grammar by periodically refine the grammar by using new pattern samples.

To support this, we need a mechanism to evaluate the performance of the pattern recognizer. Based on the architecture presented in the foregoing chapter, patterns identified by the parsers are stored in a decision log. Entries in this log is audited periodically by traders as follows:

1) Each pattern is shown to a trader and the trader determines the class of the given pattern.

2) Patterns that are concluded to be the same class by the trader and by the parser are stored in S^+ of the corresponding pattern grammar.

3) Conflicting patterns are stored in S^- of the pattern grammar.

4) If the number of conflicting patterns in a pattern grammar detected in step 3) has dropped below a threshold level, the inference procedure is invoked to derive a new grammar based on the new pattern sample.

The schematic diagram of a pattern recognizer with langauge refinement is shown in Fig. 3.18. Pattern Recognizers so constructed can be integrated with a trading expert system to provide real time data pertaining to price movement. The built-in mechanism of language refinement will automatically adjust the pattern knowledge which are

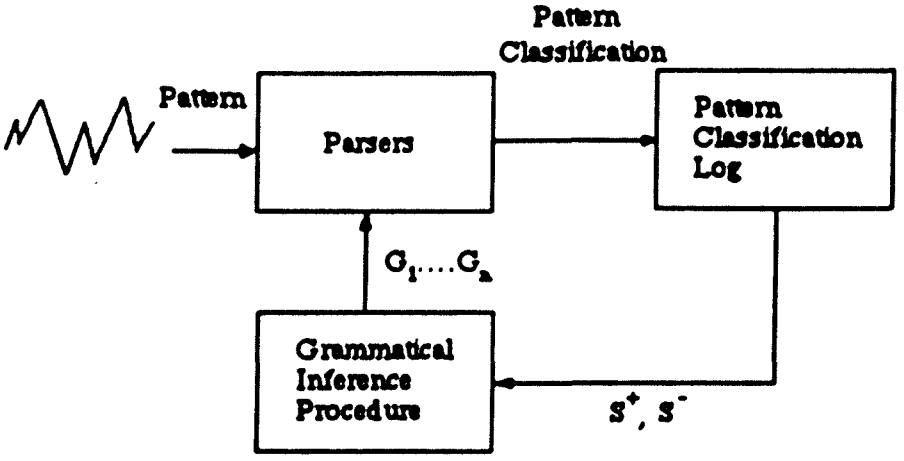

Fig. 3.18. Pattern Recognizer with Language Refinement

represented in the form of grammar rules. These refined rules are then used to build parsers for the pattern recognition engine.

Obviously, not all trading decisions involve merely the ability to recognize patterns. The assessment of the supply and demand of securities, generally referred to as fundamental analysis, is also essential in formulating trading strategies. By delegating the task of technical analysis (i.e. pattern recognition) to a computer, a trader can concentrate on assessing these factors. This allows the trader to respond more quickly to incoming market information than otherwise.

CHAPTER 4
TRADING RULES ACQUISITION AND REFINEMENT

4.1. Introduction

The world economy is experiencing an explosion in financial innovations. The result is a proliferation of financial instruments. The impact is indeed significant. New alternatives in funding, hedging, arbitraging, underwriting and investing are made possible by these innovative products. However, the complexity involved in formulating a trading strategy has also increased monotonically with these instruments. The trading process is further complicated by the fact that the market is getting more and more volatile and unpredictable. Selling and buying decisions have to be made on the spot under extremely tight time constraints.

To support decision making, complex models of the markets have been developed in an effort to provide analytical guidance to traders. However, in practice, especially in situations of high price volatility and under tight time constraints, these models offer little help. Analytical models on their own are not sufficient. Human judgments are resorted to in

these situations. In fact, the decision-making process of a trader is driven by his experience, perception, and risk perference, in most cases. This explains, at least partially, why traders respond differently under identical market conditions. Thus, it is plausible to believe that in every trader's mind there is a unique decision model associated with the markets.

Apparently, trading decisions executed by these traders are not random, but are guided by their models which evolve through years of trading practice. Cognitively, this mental model of the market is difficult to elicit. The same kind of difficulties are experienced by knowledge engineers during the knowledge acquisition process in developing expert sytems. As pointed out in [Ericsson and Simon 1984], only the information residing in short term memory can be easily articulated. Numerous techniques have been developed to assist knowledge engineers in carrying out the task in a more effective manner. These techniques go in two directions. Some of them are manual techniques such as interviewing [Newell and Simon 1972] and protocol analysis [Waterman and Newell 1971]. Others are concerned with the development of computer-based knowledge acquisition systems.

The approach described in this chapter falls into the second category. Two inductive procedures are presented to acquire trading knowledge based on a set of sample decisions. These sample decisions can be readily

obtained from previous trading transactions. Here, trading knowledge takes the form of rules called trading rules. A typical trading rule is shown below :

if the Yen/Dollar exchange rate has broken the up support

and the trade deficit between U.S. and Japan shows no sign of decline

and the budget deficit of U.S. remains high

then

take a long position in Yen Futures

A trading rule describes how a particular market condition is interpreted or reacted to by a trader. The market model of a trader is composed of a set of these trading rules. In the above rule, a trader decides to take a long position in Yen Future only if 1) the exchange rate movement has broken the up support, 2) the trade deficit shows no sign of declining, and 3) the budget deficit remains high. Thus, if the actual market situation satisfies these three conditions, then buying Yen Futures is recommended. Moreover, it is quite possible that some of these conditions are dependent on others. For example, the trade deficit between Japan and the U.S. might in turn depend on how far away from the next election year and how open is the Japanese market in the near year, as shown by the following trading rule.

if the time from next election year is more than a year

and the Japanese market is not open in the next year

then

trade deficit remains high

Each trader has his own set of trading rules which govern his decision making process. Obviously, a successful trader is one who makes the right decision at the right time. Good decisions are ones that have positive impacts on the company in monetary terms. Hence, trading rules are valuable assets of the trader and his company as well. In this chapter we will present two inductive procedures for acquiring these rules and discuss how these rules can be used in constructing intelligent trading systems. Indeed, a number of banks and brokerage houses have launched projects to develop expert systems to assist their trading activities [Reid 1986]. In order to have an expert system function in conformance with its initial specification, the right kind of knowledge (i.e. trading rules) has to be elicited.

The induction approach we use here is based on the conceptual frameork set forth by Winston [1975] Michalski [1980] [1983], Buchanan [Buchanan et. al 1980], and Mitchell [1977,1982]. A number of experimental systems which are based on this framework have been built

(AQ11 in soybean disaease diagnosis [Michalski et. al 1980a] and Meta-DENDRAL in chemical structure analysis [Buchanan et. al 1980]). Although the forms of knowledge representation used in these systems vary, the learning techniques used are collectively referred to as "learning from examples" (or "learning by induction") [Dietterich et al. 1983]. Acquiring knowledge by using these techniques has proved to be a feasible alternative to circumvent the bottleneck of knowledge acquistion in building expert systems. In fact, it was claimed that AQ11's result in soybean disease diagonsis has outperformed experts in some cases [Michalski et al.1980a, 1980b].

The appealing results obtained from some of these projects have revealed the potential application of induction as a general knowledge acquisition paradigm. We proceed by first presenting in Section 4.2. a rule-based language that is used to state trading rules. Three spaces are then derived from the language, allowing one to envision the induction operators/procedures from a state space perspective. Section 4.3. sketches out the basic idea of the two procedures and explains the mechanism by using the notion of space partitioning. Section 4.4. introduces a number of induction operators which are used to transform trading rules. The two

procedures are presented in Section 4.5. and illustrated with examples. Section 4.6. concludes with a discussion of knowledge refinement.

4.2. A Language for Trading Rules

In this section, trading rules are stated using a rule langauge defined below. One form of knowledge representation scheme commonly used in building expert systems is production rules. The rule language described here can be considered as a specific form of production rules. There are three syntactic constructs in this language : Feature, Template, and Rule.

Features -- A feature represents an independent attribute of a market. These features, F_1,F_2, ..., F_n where n is finite, together identify the dimensions of the market.

e.g., Interest-Rate, Economy-Status

Features Domains -- For each feature, there is a finite set of values associated with it. They are called feature domains and are denoted by $Dom(F_1)$, $Dom(F_2)$, .. $Dom(F_n)$. Each domain specifies the scope of a feature, and collectively, they specify the scope of the market.

Furthermore, elements of a domain set can either be ordered or unordered.

e.g., Dom(Economy-Status) = {Good, Poor, Fair} is defined to be unordered and Dom(Interest-Rate) = {0.0, 0.1, 0.2,...200.0} is ordered

Ordering of domain values is important because some learning operators can only be applied to ordered feature domains. Learning operators will be explained in detail in Section 4.4.

Templates - A tuple of the form $(F_i\ V_i)$ where $Dom(F_i) \supseteq V_i$, $1 \leq i \leq n$. A template identifies the specific value(s) associated with a feature.

e.g., (Interest-Rate 7.8)

Rules - A rule consists of two parts : condition and conclusion separated by "=>". It takes the form

$$(F_1\ V_1)(F_2\ V_2)...(F_{m-1}\ V_{m-1}) \Rightarrow (F_m\ V_m),$$

where $Dom(F_i) \supseteq V_i$, i=1,2,..m.

The condition part consists of a list of templates. The ordering of templates in a condition is not important. In logical terms, one can think of the template list as a conjunctive statement without variables. As will be

discussed later, a condition characterizes a subspace which cannot be altered by the order of the templates.

e.g., (Interest-Rate 7.8)(Unemployment-Rate ≤ 5.6) => (Economy-Status good)

The semantics of a rule as illustrated by the above example is that the condition part specifies the current situation (or interpretation) of the market (i.e. Interest-Rate is 7.8% and Unemployment-Rate is less than 5.6%). Based on this condition, the conclusion (i.e. Economy-Status is good) is drawn. The less than sign "≤" is a shorthand for a subset of ordered domain values. Whether (Interest-Rate 7.8) proceeds (Unemployment-Rate ≤ 5.6) or the other way around is not important and does not alter the meaning of the rule.

Based on the above language, three spaces are defined as follows :

1) Feature Space F

$$F = Dom(F_1) \times Dom(F_2) \times ...Dom(F_n)$$

F is the Cartesian product of individual feature domains. It defines the scope of the market and thus covers all possible market situations. An element $x \in F$ is called a state in F.

2) Template Space T

$$T = \{F_1 \times Dom(F_1)\} \cup \{F_2 \times Dom(F_2)\} \ldots \cup \{F_n \times Dom(F_n)\}$$

T is the set of templates given F. The power set of T minus $\{\phi\}$ is denoted by P(T). P(T) is the set that contains all subsets of T, excluding the empty set. From a state space perspective, an element of P(T) divides F into two disjointed partitions. To illustrate this, let us consider an element of P(T), say t, where $t = \{(F_1\ V_1)\ (F_2\ V_2)\}$. The partitions induced by t on F are

$$t' = \{Dom(F_1) - V_1\} \times \{Dom(F_2) - V_2\} \times Dom(F_3) \ldots \times Dom(F_n)$$

$$t'' = V_1 \times V_2 \times Dom(F_3) \times Dom(F_4) \ldots \times Dom(F_n)$$

Note that $t' \cup t'' = F$ and $t' \cap t'' = \phi$. F are divided by t into 2 disjointed partitions : elements in F with values of F_1 and F_2 that cover V_1 and V_2, respectively, are contained in t'', with t' containing the remaining elements.

3) Rule Space R

$$R = \{\ z \mid z : x \Rightarrow y \text{ where } x \in P(T) \text{ and } y \in T\ \}$$

In the context of security trading, R is interpreted as the set of all possible trading rules that a trader uses to relate market conditions to conclusions.

4.3. A Sketch of the Induction Procedures

Let us sketch out the mechanism of the induction procedures by using the language defined above. By induction, we mean to infer general principles from specific instances. In this chapter, this is translated to inferring trading rules from previous trading decisions. Given the above language, there is conceptually no difference between a rule and an example. Both are partitions of F. In the following discussion, the terms "rules," "example," and "partitions" can be interchanged. To illustrate this, let us consider an example (Fig. 4.1.) of buying Yen Futures as shown below:

(+ve) (Trade-Deficit large)(Japan-Prime-Rate high) => (Buy Yen-Future)

For simplicity, only two features are used in this example. For the rest of the chapter, we use (+ve) and (-ve) to denote positive and negative examples, respectively. These two features (Trade-Deficit and Japan-Prime-Rate) and their associated values induce a partition in F. This was explained in the previous section. In a similar fashion, a negative example of buying Yen Futures such as

(-ve) Japan-Prime-Rate low) => (Buy Yen-Future)

is also a partition in F (Fig. 4.2.)

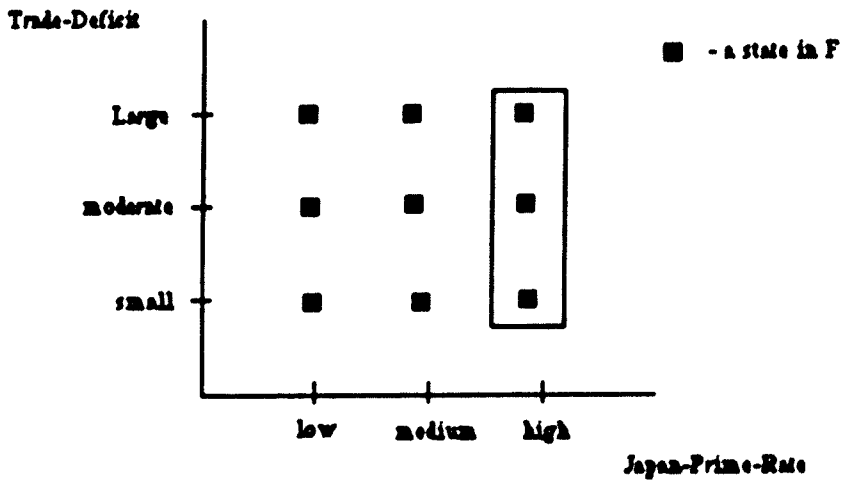

Fig. 4.1. Partition of (Trade-Deficit large)(Japan-Prime-Rate high) =>
(Buy Yen-Future)

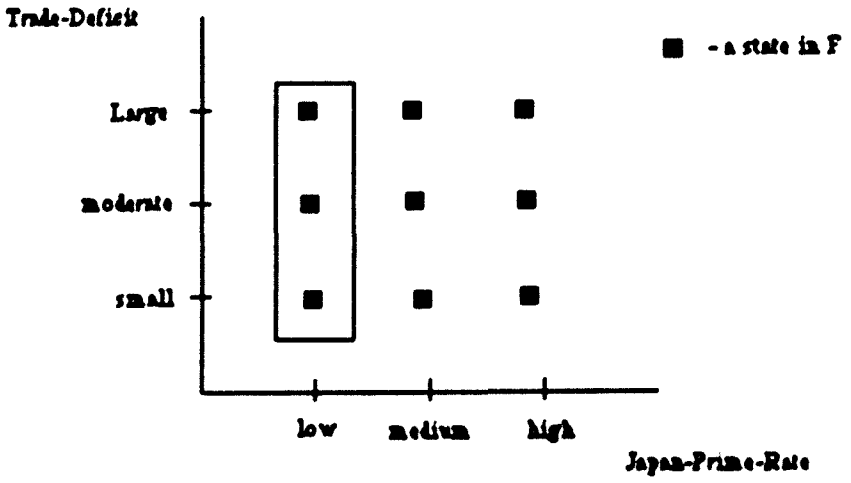

Fig. 4.2. Partition of (Japan-Prime-Rate low) => (Buy Yen-Future)

Given a collection of examples of a decision, say buying Yen Futures, the induction procedures attempt to infer the relationships between the market situations and the decision of buying Yen Futures from these examples. Each example describes a unique market condition that relates to a decision of buying Yen Futures. Negative examples are also used. A negative example spells out the wrong market condition related to a decision. Similarly, negative examples appear as partitions in F. The induction procedures discussed in this section are search methods that search for partitions in F (or equivalently rules in R) that are consistent with the given examples.

The idea can be explained pictorally in Fig. 4.3. Given a collection of initial partitions (both positive and negative examples of a decision), an induction procedure expands, contracts, and unifies these partitions so as to generate a new set of partitions (rules) that are more general (larger) than the initial examples. The ideal case here is to partition F in such a way that positive and negative examples reside in different partitions. The operators required to alter the partitions and how these operators are used in the induction procedures are discussed in the next two sections.

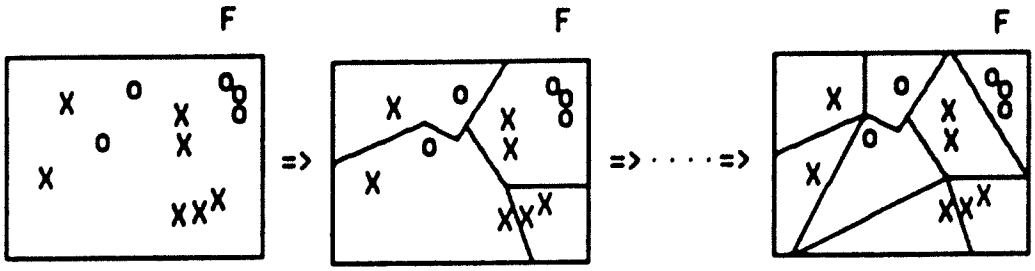

Fig. 4.3. Rules Induction

4.4. Induction Operators

Induction operators are the means by which knowledge (ie. trading rules) can be transformed. As indicated in surveys by Bunley et al. [1985] and Dietterich et al. [1983], there are basically two types of operators in most rule acquisition systems : generalization operators and specialization operators. Generalization and specialization operators generalize and specialize rules, respectively. In terms of partitioning, the former expand and unify partitions while the latter contract partitions. Some of these operators are context sensitive in the sense that they can only be applied to ordered feature domains. The induction operators associated with the rule language are listed below:

Generalization Operators :

The sign "G>" means "generalizes to"

1) **Dropping-Conditions**

Dropping condition operator generalizes a rule by dropping a template from the condition of a rule.

$$(F_1\ V_1)(F_2\ V_2) \Rightarrow (F_3\ V_3)$$

$$G>$$

$$(F_1\ V_1) \Rightarrow (F_3\ V_3) \quad \text{or} \quad (F_2\ V_2) \Rightarrow (F_3\ V_3)$$

eg. (Trade-Deficit large)(Japan-Interest-Rate high) => (Buy Yen-Future)

G>

(Trade-Deficit Large) => (Buy Yen-Future) or

(Japan-Interest-Rate high) => (Buy Yen-Future)

2) **Min-Max-Reference**

Min-Max-Reference operator is applicable to rules that have the same feature on their conditions but are associated with different singleton values. These singleton values are then expanded to sets of values by taking the minimum and maximum of V_1 and V_2, respectively. Note that in order to apply this operator, the feature domain has to be ordered.

$(F_1\ V_1) \Rightarrow (F_3\ V_3)$

G> $(F_1 \leq \max(V_1, V_2)) \Rightarrow (F_3\ V_3)$ or

$(F_1 \geq \min(V_1, V_2)) \Rightarrow (F_3\ V_3)$

$(F_1\ V_2) \Rightarrow (F_3\ V_3)$

where $Dom(F_1)$ is ordered

For example, (Unemployment-Rate 9.8) => (Economy-Status poor) and

(Unempolyment-Rate 11.0) => (Economy-Status poor)

G>

(Unemployment-Rate ≤ 11.0) => (Economy-Status poor) or

(Unemployment-Rate ≥ 9.8) => (Economy-Status poor)

where Dom(Unemployment-Rate) = {0.0,0.1,.....50.0}

3) **Extending-Reference**

Similar to the Min-Max-Reference, this operator applies to rules that have the same feature in their conditions. However, the Extending-Reference operator is applicable not only to singleton values, but to set of values as well. It unifies the different set of values associated with the feature.

$(F_1\ V_1) \Rightarrow (F_3\ V_3)$

$$G> \qquad (F_1\ V_1 \cup V_2) \Rightarrow (F_3\ V_3)$$

$(F_1\ V_2) \Rightarrow (F_3\ V_3)$

For example,

(Budget-Deficit large => (Treasury-Bill-Interest-Rate-Auction up) and

(Budget-Deficit moderate) => (Treasury-Bill-Interest-Rate-Auction up)

G>

(Budget-Deficit {large,moderate})=>(Treasury-Bill_interest-Rate-Auction up)

4) Next-High-Point

This operator extends the set of values associated with a feature by including the next higher element into the set.

$(F_1 \leq V_1) \Rightarrow (F_3\ V_3)$ **G>** $(F_1 \leq V_2) \Rightarrow (F_3\ V_3)$

where Dom(F_1) is ordered and V_2 is the next higher value of V_1

For example, (Unemployment-Rate ≤ 11.0) => (Economy-Status poor)

G>

(Unemployment-Rate ≤ 11.1) => (Economy-Status poor)

where Dom(Unemploymenet-Rate) = {0.0,0.1,..11.0,11.1,...50.0}

5) Next-Low-Point

In a similar fashion, Next-Low-Point generalizes a rule by including the next lower value in the set of values associated with a feature.

$(F_1 \geq V_2) \Rightarrow (F_3\ V_3)$ **G>** $(F_1 \geq V_1) \Rightarrow (F_3\ V_3)$

where Dom(F_1) is ordered and V_2 is the next higher value of V_1

For example, (Unemployment-rate ≥ 11.1) => (Economy-Status poor)

G>

(Unemployment-Rate ≥ 11.0) => (Economy-Status poor)

where Dom(Unemployment-Rate) = {0.0,0.1,..11.0,11.1,..50.0}

Specialization operators :

the sign "S>" means "specializes to"

1) **Adding-Conditions**

This operator is the reverse of the Dropping-Condition generalization operator. It combines two rules by combining the conditions of the two rules into one.

$(F_1\ V_1) \Rightarrow (F_3\ V_3)$

$$S> \quad (F_1\ V_1)(F_2\ V_2) \Rightarrow (F_3\ V_3)$$

$(F_2\ V_2) \Rightarrow (F_3\ V_3)$

For example, (Unemployment-Rate 15.0) => (Economy-Status poor) and

(Inflation-Rate 17.0) => (Economy-Status poor)

S>

(Unemployment-Rate 15.0)(Inflation-Rate 17.0) => (Economy-Status poor)

where Dom(Unemployment-Rate) = {0.0,0.1,..11.0,11.1,..50.0},

Dom(Inflation-Rate) = {0.0,0.2,...15.4,15.6,....300.8}

2) **Clos.ng-Interval**

When this operator is applied, the range of values associated with a feature is bounded from either above or below by the next higher or lower

value, respectively. Note that this operator is applicable to ordered feature domains only.

$$(F_1 \geq V_1) \Rightarrow (F_3\ V_3) \quad S> \quad (F_1\ \{V_1,V_2\}) \Rightarrow (F_3\ V_3)$$

$$(F_1 \leq V_2) \Rightarrow (F_3\ V_3) \quad S> \quad (F_1\ \{V_1,V_2\}) \Rightarrow (F_3\ V_3)$$

where $Dom(F_1)$ is ordered and V_2 is the next higher value of V_1

For example,

(Inflation-Rate ≥ 15.4) => (Economy-Status poor)

S>

(Inflation-Rate {15.4,15.6}) => (Economy-Status poor)

where Dom(Inflation-Rate) = {0.0,0.2,...15.4,15.6,....300.8}

3) **Next-High-Point**

The Next-High-Point specialization operator reduces the set of ordered values of a feature by discarding the smallest value in the set.

$$(F_1 \geq V_1) \Rightarrow (F_3\ V_3) \quad S> \quad (F_1 \geq V_2) \Rightarrow (F_3\ V_3)$$

where $Dom(F_1)$ is ordered and V_2 is the next higher value of V_1.

For example,

(Persian-Gulf-Tension ≥ tight) => (Gold-Future-Trend up)

S>

(Persian-Gulf-Tension extremely-tight) => (Gold-Future-Trend up)

where Dom(Persian-Gulf-Tension) = {stable,tight, extremely-tight} and is ordered.

4) **Next-Low-Point**

Next-Low-Point reduces the set of ordered values of a feature by discarding the largest value from the set.

$$(F_1 \leq V_2) \Rightarrow (F_3\ V_3) \quad S> \quad (F_1 \leq V_1) \Rightarrow (F_3\ V_3)$$

where Dom(F_1) is ordered and V_2 is the next higher value of V_1.

For example,

(Inflation-Rate ≤ 5.6) => (Economy-Status fair)

S>

(Inflation-Rate ≤ 5.4) => (Economy-Status fair)

where Dom(Inflation-Rate) = {0.0,0.2,...15.4,15.6,....300.8}.

One can think about these operators as functions that map rule(s) to rule. By application of these two kinds of induction operators, the number of states in F covered by a rule can be altered. An induction procedure is an algorithm that systematically applies these operators to partition F to generate rules that satisfy certain predefined induction criteria. The two

induction procedures, one based on specialization operators and the other on generalization operators, will be presented in the next section.

4.5. Two Induction Procedures based on Space Partitioning

Both of these procedures are centered on the concept of "learning from examples" [Dietterich et al. 1982]. Their input-output requirements are depicted in Fig 4.4. To learn a concept or a trading decision, previous scenarios or cases are used as examples from which general trading rules are inferred.

Another parameter of these two procedures consists of the induction criteria. They characterize the goal of the search procedure by deciding which search space should be pruned. In a survey by Angluin et al. [1983] on inductive inference, the majority of practical and theoretical studies on inductive inference methods are associated with two conflicting criteria - Simplicity and Goodness of Fit.

Simplicity - In characterizing a concept, a simple yet powerful description is most desired. In other words, we would prefer a rule that has the smallest number of features in its condition. Suppose we have the following two rules relating to Economy-Status :

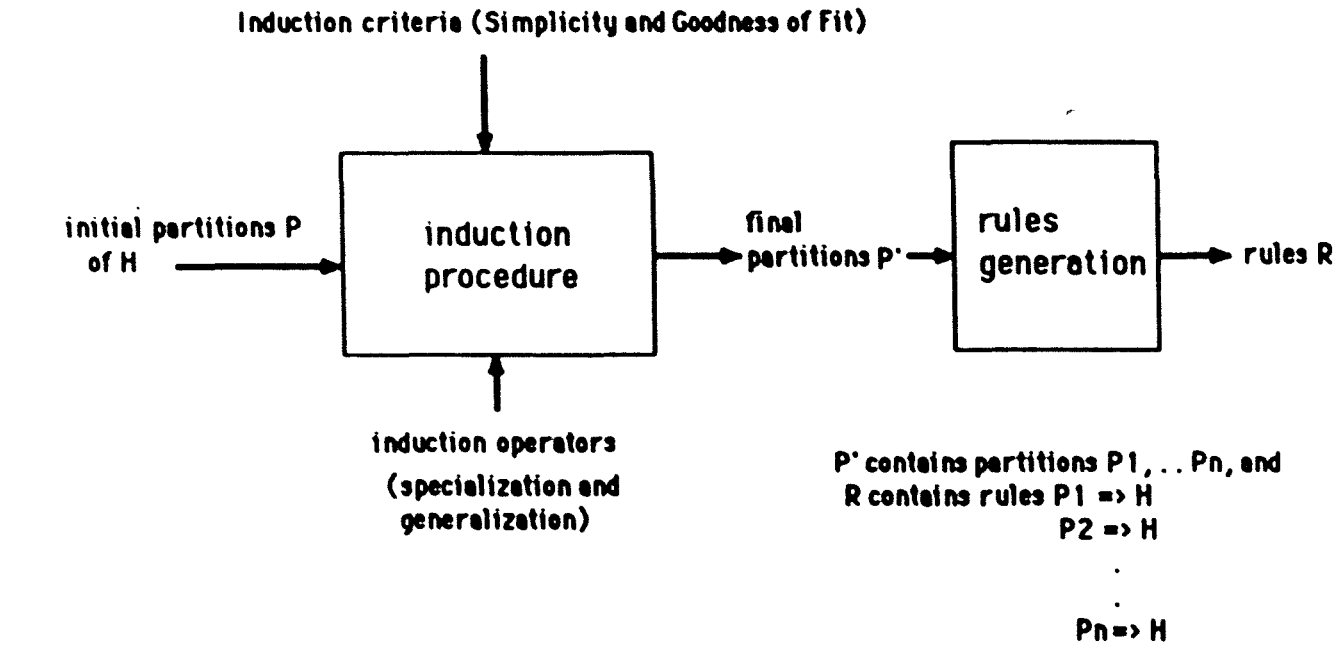

Fig. 4.4. Induction Procedure Based on Space Partitioning

1) (Inflation-Rate ≤ 4.6)(Unemployment-Rate ≤ 10.0) =>

(Economy-Status good)

2) (Trade-Surplus high) => (Economy-Status good)

If both rules cover all positive examples and reject all negative examples of a good economy, then rule 2 is preferred over rule 1. The reason is that Trade-Surplus is the dominant feature in describing a good economy, and having only one feature does not impede the validity of rule 2 in describing a good economy.

Goodness of fit - In the search for the simplest description of a concept, the result rule might consist of very few features. Yet, it might be too general to apply in other situations. For instance, the statement "all good students are human beings" is too general in describing good students. In order to prevent a concept from being over generalized, the generalization process should be driven by the examples provided. The idea is similar to regression analysis in statistics in which a linear regression line that minimizes the least square distances from the provided data points is estimated.

In our procedures, we would prefer a rule that partition F in such a way that positive and negative examples are totally separated. A rule would probably become more complicated (involved more features) in order to cover more positive examples (or reject more negative examples). Moreover, it might not be possible to have a single rule in cases when examples are not clustered in a local region in F. Simplicity and Goodness of fit are two conflicting criteria that require tradeoffs to be made and input to the induction procedures. For instance, Fig. 4.5. is a list of combinations between the two criteria which have no absolute preference ordering. The tradeoff between these two criteria is specified by five parameters in the procedures :

w_1 - number of positive examples covered by a rule (local)

w_2 - number of negative examples rejected by a rule (local)

w_3 - number of features in a rule

w_4 - number of positive examples covered by a set of rules (global)

w_5 - number of negative examples rejected by a set of rules (global)

A distinction is made here between the number of examples covered (or rejected) by a rule and that by a set of rules. The former is referred to as local while the latter global. The reason is that it might require more

	no. of +ve examples covered	no. of -ve examples rejected	no. of features
A	12	26	5
B	12	16	4
C	8	18	4
D	14	26	6

Fig. 4.5. Four Combinations of Criteria with No Absolute Ordering

than one rule in relating market conditions to a conclusion in order to be consistent with the examples. As mentioned before, examples that scatter in F might form clusters which cannot be described by a single rule. Instead, they are described by a set of rules. Therefore, separate parameters w_4 and w_5 are required when rules are assessed collectively.

For instance in Fig. 4.6., there are 2 positive examples and two negative ones. Suppose we specify that each rule has to cover 50% (w_1) of positive examples and reject 50% (w_2) of negative examples. The two rules are satisfied individually. However, if the two rules are considered jointly, they together cover 100% of positive examples and reject 0% of negative examples !

Notice that the actual local acceptance rate of positive examples is always greater than or equal to the actual global rate. But this is not true for negative examples. The actual local rejection rate always equals or underestimates the global one as illustrated in the above example. In order to have a feasible set of criteria, we would like to specify values of w_1, w_2, w_4 and w_5 that satisfy the following constraints :

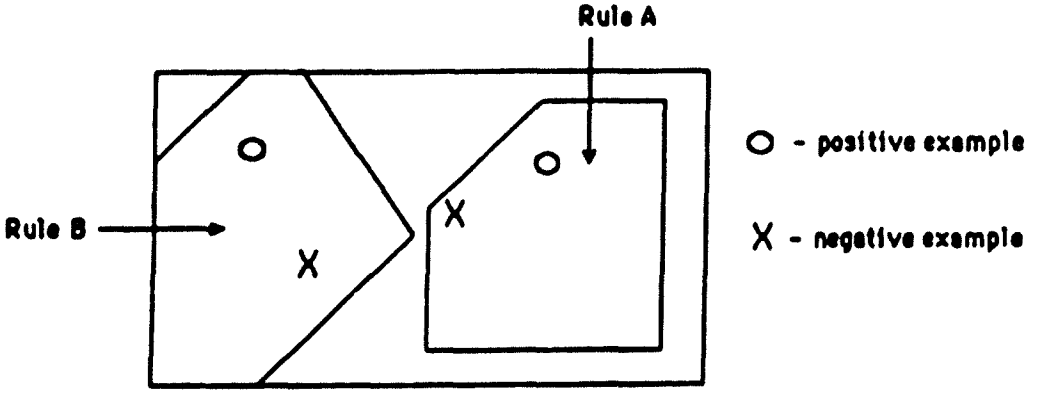

Fig. 4.6. Example of Local and Global Criteria

1) $w_1 \leq w_4$

2) $w_5 \leq w_2$

The proof is as follows : Suppose n rules, $R_1, R_2, ..., R_n$, are used to describe a trading decision. Given a set of positive examples N_+ and a set of negative examples N_- of this decision, let $P_+(R_i)$ and $P_-(R_i)$, $1 \leq i \leq n$, denote the actual number of positive examples and negative examples covered by each of these rules, respectively. It follows that the positive examples covered by these rules is

$$P_+(R_1) \cup P_+(R_2) ..... \cup P_+(R_n) \supseteq P_+(R_i), 1 \leq i \leq n \qquad (4\text{-}1)$$

and the global acceptance rate of positive examples become

$$| P_+(R_1) \cup P_+(R_2) ..... \cup P_+(R_n) | / |N_+| \geq | P_+(R_i) | / |N_+|, 1 \leq i \leq n \qquad (4\text{-}2)$$

In (4-2), L.H.S of the inequality represents the actual global acceptance rate of positive examples, and R.H.S. of (4-2) denotes the acceptance rate of individual rules, implying that $w_4 \geq w_1$. Similarily, the negative examples rejected by these rules is

$$N_- - P_-(R_i) \supseteq N_- - P_-(R_1) \cup P_-(R_2) ..... \cup P_-(R_n), \quad 1 \leq i \leq n \qquad (4\text{-}3)$$

The global rejection rate of negative examples then becomes

$$|\{N_- - P_-(R_i)\}| / |N_-| \geq |N_- - P_-(R_1) \cup P_-(R_2) ..... \cup P_-(R_n)| / |N_-| \qquad (4\text{-}4)$$

where $1 \leq i \leq n$

Since the L.H.S. of (4-4) is the actual rejection rate which is always greater than or equal to the local rejection rate (R.H.S. of 4-4), in order to be feasible, we need $w_5 \leq w_2$.

Let us proceed to discuss the algorithm of the two procedures. We start with the one that uses specialization operators only.

Induction Procedure (Specialization)

1. initialize final rule set R to { } and raw rule set R' to F;
2. initialize R' with feature domain constraints if any;
3. **while** R' ≠ { } **do**
4. generate new rules by applying specialization operators to rules in R';
5. **for** each raw rule r generated in 4 **do**
6. **if** r covers at least w_1% positive instances and the number of .eatures ≤ w_3 **then begin**
7. R' = R' ∪ {r};
8. R' = R' - parent(s) of r

9. **End**(if);

10. **for each** parent rule P_r in R' **do**

11. **if** no successor rule is in R' **then begin**

12. $R = R \cup \{P_r\}$;

13. $R' = R' - \{P_r\}$

14. **End**(if);

15. **End** (while);

16. Remove all rules in R which reject less than w_2% of negative examples;

17. Check rules in R against the global parameters w_4 and w_5;

18. Remove redundant rules in R';

The procedure is a general-to-specific breadth-first search procedure that generates rules by incrementally specializing F or the initial partitions of F. It operates on two rule sets R and R'. R contains final rules while R' contains candidate rules to be specialized. Initially (step 1), R is null - no final rule has been generated. If additional information or constraints are available, they can be incorporated into the procedure in the form of initial partitions of F. This takes place in step 2. This information can be obtained from a number of sources (experts and

observations) and is useful in reducing the time of search. For instance, if the Inflation-Rate is known to be less than 10% when the economy is good ((Inflation-Rate ≤ 10.0) => (Economy-Status good)), then the search procedure can neglect those elements in F with Inflation-Rate larger than 10%.

Depending on whether extra information is available or not, R' is initialized to F or the initial partitions of F accordingly. The search procedure starts in step 3 : for each rule in R', specialization operators are applied to generate more specialized rules. After all rules in R' are tried, a new set of rules which are more specialized than their parents has been created. Duplicated or redundant rules are discarded from R'. Since this is a general-to-specific search procedure by incrementally specializing F to fit the examples, if there exist two rules in R' with one implying the other, then the more general one is discarded. The procedure then checks whether each new rule covers the prescribed percentage of positive examples (w_1) and the number of features (w_3). If satisfied, the new rule is put into R' and the parent of which is deleted from R'. The reason is that since the child is more specific than its parent(s) and satisfies w_1 and w_3, there is no need to keep the parent(s) in R'. These operations take place in

steps 5 - 9. Note that rules in R' always satisfy w_1 and w_3 during the search.

There might be cases that some rules in R' do not have children that satisfy both w_1 and w_3. These rules are then discarded from R' and are put into R (steps 10-14). Steps 4 - 14 are then repeated until R' is empty. When R' becomes empty, it implies that no rule can be specialized anymore. If no rule is collected in R after R' becomes null, then no rule can satisfy the given w_1 and w_3 values. The procedure has to be rerun with different w_1,w_2 and w_3 values. In cases when R is not null, the procedure will retain those rules in R that also satisfy w_2. Finally, the set of rules in R is checked with w_4 and w_5 on a global basis. This is carried out by counting the number of positive examples that are covered by rules in R. Negative examples are done in a similar fashion. If satisfied, R is output as the final rule set. Otherwise, the procedure has to be rerun with different parameter values. In the last step, redundant rules are eliminated. The search r echanism of the procedure can be illustrated by the following example.

Example

In this hypothetical example, we want to learn the trading rules of buying Yen Futures. Four previous trading decisions are used as input examples to the procedure. Out of the four examples, two are positive and two are negative. To further simplify these examples, only two features are used in the four examples as defined below :

TD - Trade deficit between Japan and U.S. (in billions $)

IR - (1 + one year bond rate of Japan)/(1 + one year bond rate of U.S.) ,

where

Dom(TD) = {0... 50,60,70,...200} (ordered)

Dom(IR) = {0.200,0.201,.....5.000} (ordered)

(+ve/-ve)	Rules
+ve	(TD 70)(IR 1.098) => (Buy Yen-Future)
+ve	(TD 63)(IR 1.095) => (Buy Yen-Future)
-ve	(TD 20)(IR 1.080) => (Buy Yen-Future)
-ve	(TD 10)(IR 1.001) => (Buy Yen-Future)

If we are provided with the initial beliefs that it is the right condition to buy Yen Future whenever $TD \geq 50$ or $IR \leq 1.100$, then our

procedure can start from the partitions induced by the following two rules:

1. (TD $\geq$ 50) => (Buy Yen-Future)

2. (IR $\leq$ 1.100) => (Buy Yen-Future)

In this example, w_1,w_2,w_3,w_4,and w_5 are set to 100, 100, 3, 100, 100 respectively. The entire search procedure of this example is shown in Fig.4.7.. The final rule obtained in R is (TD {60,70})(IR $\leq$ 1.098) => (Buy Yen-Future).

Induction Procedure (Generalization)

The difference between this procedure and the previous one is that it uses a specific-to-general search technique as opposed to the general-to-specific approach in the previous procedure. It starts with an initial set of positive examples and incrementally generalizes the positive examples. The idea is to generalize some specific cases to obtain more general rules. The search mechanism is similar to Michalski' INDUCE method in [Michalski 1983] in the way that positive examples serve as the seeds of generalization.

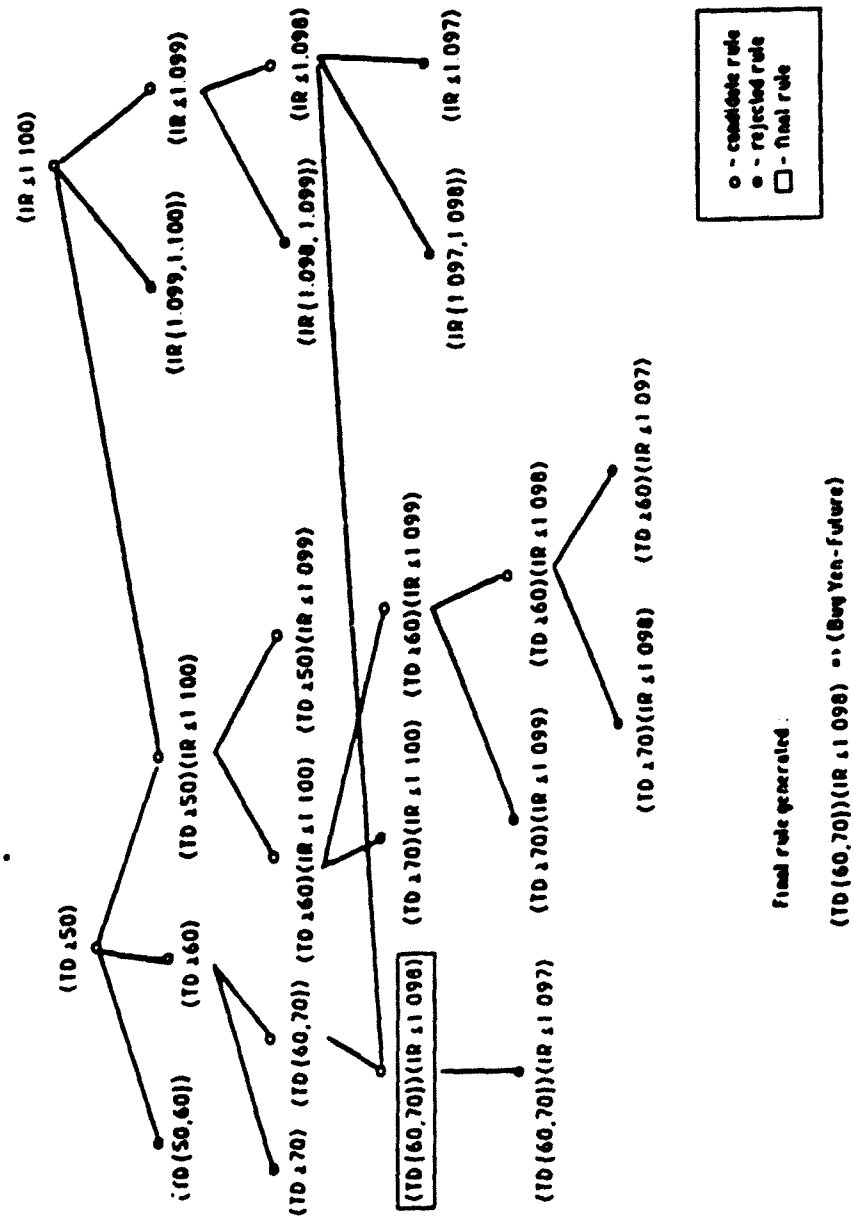

Fig. 4.7. The Search Tree of the Specialization Procedure - An Example

1. initialize final rule set R to { };
2. initialize raw rule set R' with positive examples and constraints if any;
3. **while** R' ≠ { } **do**
4. generate new rules by applying generalization operators;
5. **for** each new rule r generated in step 4 **do**
6. **if** r rejects at least w_2% of negative examples and contains less than w_3 features **then**
7. $R' = R' \cup \{r\}$
8. **end**;(for)
9. **for** each parent rule P_r in R' **do**
10. **if** no child is in R' **then**
11. $R = R \cup \{P_r\}$;
12. $R' = R' - \{P_r\}$
13. **End**;(for)
14. **End**;(while)
15. **For** each rule r in R **do**
16. **if** r covers less than w_1% of positive examples **then**
17. $R = R - \{r\}$;

18. **End**;(for)

19. **check if each rule in R satisfies the global parameters w_4 and w_5;**

20. **discard redundant rules in R.**

Steps 1 and 2 initialize the two rule sets R and R' with { } and positive examples, respectively. Generalization operators are then applied to rules in R' to obtain more generalized ones (Step 4). Only those new rules with less than w_3 number of features which also reject w_2% of negative examples are retained in R' (Step 5 - 8). Again, duplicated or redundant rules are discarded from R'. Rules that are covered by others are redundant and are discarded from R'.

The next step is to identify rules in R' that cannot be generalized anymore. They are those rules with all their children rejected in step 6. These rules are discarded from R' and put into the final rule set R. All other parent rules are also eliminated from R because the newly generated rules are more general and their parents need not be retained. The above operations are repeated until no more rules can be generalized (ie. R'= { }).

The rules obtained in R after the procedure exists from the main search loop are tested whether they cover w_1% of positive examples or not (Step

15-18). Rules that fail the test are discarded from R. If R is null, the search is concluded to have failed with the given w_2 and w_3 values. The search has to be restarted with another set of parameters. In step 19, the remaining rules are then tested globally against the specified w_4 and w_5 values. Finally, redundant rules are deleted from the final rule set. If satisfied, R is output as the final rule set. Otherwise, the procedure has to be rerun again with a different set of parameter values.

Example

Fig. 4.8. illustrates the search mechanism of the generalization procedure as applied to the previous example. The same set of feature names is used. Four different examples are used as shown below :

(+ve/-ve)	Rules
+ve	(TD 70)(IR 1.090) => (Buy Yen-Future)
+ve	(TD 80) => (Buy Yen-Future)
-ve	(TD 65) => (Buy Yen-Future)
-ve	(IR ≤1.087) => (Buy Yen-Future)

Here, w_1, w_2, w_3, w_4 and w_5 are set to 100, 100, 3, 100, 100 respectively.

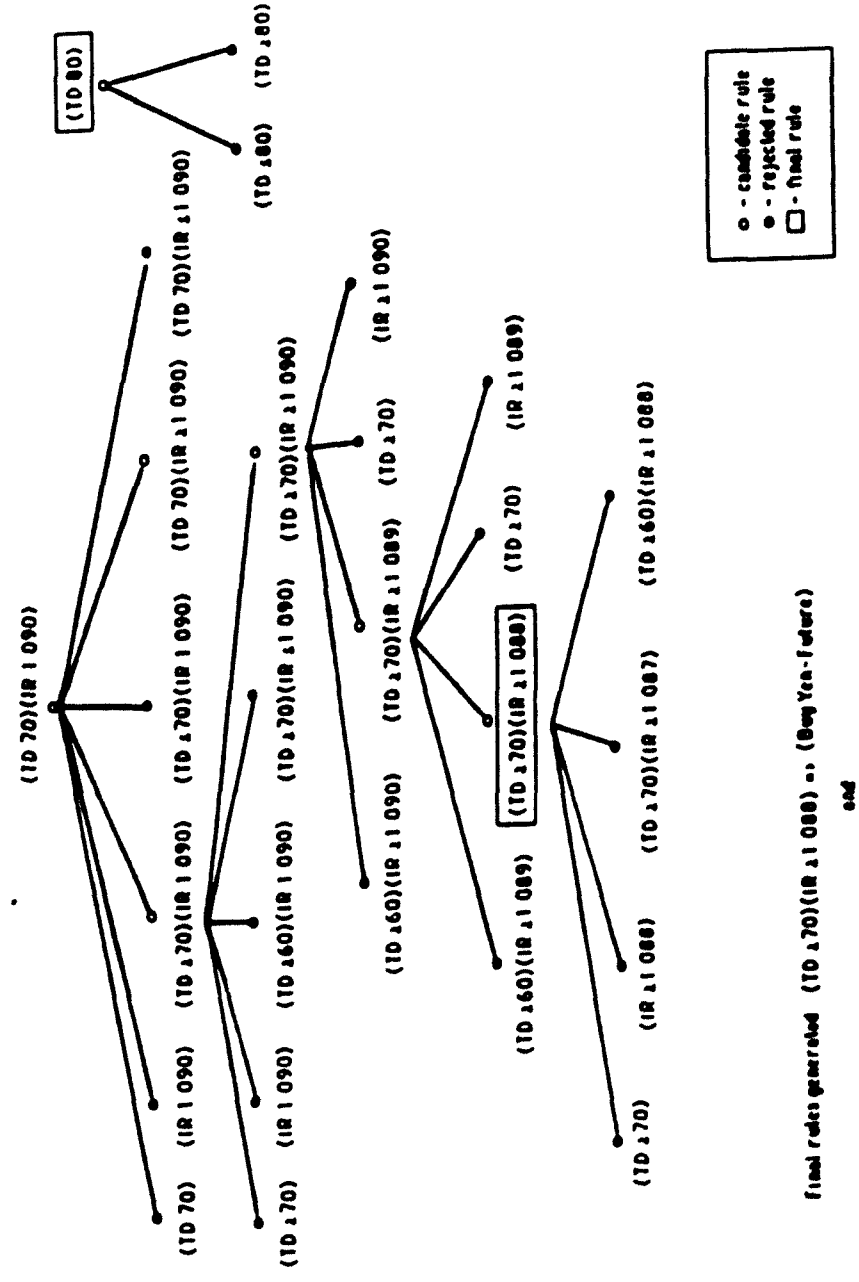

Fig. 4.8. The Search Tree of the Generalization Procedure - An Example

The final rules generated are

1. (TD ≥ 70)(IR ≥ 1.088) => (Buy Yen-Future)

2. (TD 80) => (Buy Yen-Future)

4.6. Knowledge Refinement

We have presented two procedures, both based on induction learning, in acquiring trading rules. The trading rules so acquired using these methods might become obsolete as the market envirnoment changes. To keep our knowledge updated, we have to update the rules periodically. This can take place in two ways :

1) Redefine the feature space -- new features may be added and existing features deleted from F. The size of existing feature domains may need to be enlarged or reduced. The scope of the market defined by F has to be updated in response to changes in the market environment.

2) New set of trading rules -- a new set of trading rules has to be inferred periodically to keep track of the changes in the trader's perception of the market.

The knowledge base of most current expert systems is static in nature. There is little or no mechanism provided to update the knowledge. Except for domains that do not change rapidly, the knowledge acquired might no

longer be valid at a later stage. This is particularly true in security trading where the knowledge is intensive and is changing rapidly. To build expert systems in this area necessitates an automated knowledge acquisition and updating mechanism to be built together with the inference engine. Using the induction procedures, a new set of trading rules can be generated by running the procedures with a new set of examples, assuming that the new examples fairly reflect the changes of the market.

To achieve this, we need an updating mechanism that can refine the rule base periodically. Again, the architecture in Chapter 2 is adopted here as follows :

1) Decisions generated by the inference engine, together with the corresponding inference chains, are stored in a decision log.

2) The decision log is audited periodically by human traders to check for conflicting decisions.

3) Conflicting decisions are recorded and stored as negative examples of the corresponding rules.

4) If the number of conflicting conclusions has fallen below a given threshold level, the induction procedures discussed in Section 4.5. are invoked.

CHAPTER 5
CONCLUDING REMARKS

5.1. Summary

We have extended the conventional architecture of expert systems to incorporate knowledge acquisition and refinement in this thesis. Two new components, decision log and knowledge refiner, are added. The former keeps track of the decision generated by the expert system and the corresponding chain of reasoning. By periodically audit the entries of the decision log and compare them with that of human experts, one can infer the validity of the knowledge base. The validity of a knowledge base is measured by the number of conflicting decisions between the expert system and human experts. If this number fell below a threshold value, the knowledge refiner is invoked to refine the knowledge base in light of the conflicting decisions.

The architecture is not restricted to any specific kind of knowledge representation schema or inference procedures. To demonstrate its generality, two task specific reasoning systems are studied using this extended architecture. The two tasks are pattern recognition and

classification. Furthermore, to illustrate the applicability of this architecture to solve real-life problems, both tasks are applied in automating security trading. Fig. 5.1. summaries the knowledge scheme, the inference method, the decision log, and the knowledge refining method of these two tasks.

5.2. Future Research Directions

As mentioned earlier in this thesis, the assumption that underlies this architecture is the validity of the inference method. By assuming the inference method of an expert system is consistent with the reasoning process of human experts, we are able to focus primarily on the validity of the stored knowledge. Furthermore, by using the number of conflicting decisions as the performance criterion, one cannot distinct the situations between a deteriotating knowledge base that due to a changing domain or one that due to bias sample.

This research will continue in three directions as follows :

1) To understand the changing inference process of human experts and to develop methodology to detect and monitor these changes.

	Pattern Recognition	Classification
Knowledge Scheme	Phrase Structure Grammar	Classification Rule
Inference Mechanism	Parsing	Forward/ Backward Chaining
Decision Log	Pattern Classifications	Decision and Reasoning Chain
Knowledge Refinement	Grammatical Inference	Concept Induction

Fig. 5.1. Components of Pattern Recognition and Classification Systems

2) To develop new measure of the validity of a knowledge base that can distinct between an outdated knowledge base and one that due to bias sample.

3) To test the applicability of the proposed architecture in different application domains and different reasoning tasks.

LIST OF REFERENCES

LIST OF REFERENCES

Aho, A. V. and Ullman, J. D., "The Theory of Languages," *J. Math. Syst. Theory*, 2, 1968.

Aho, A. V. and Peterson, T. G., "A Minimum Distance Error-Correcting Parsers for Context-Free Languages," *SIAM J. Comp.*, 4, 1972.

Angluin, D. and Smith, C. H., "Inductive Inference : Theory and Methods," *Computing Survey*, 15,3, 1983.

Bainbridge, L., "Asking Questions and accessing knowledge," *Future Computing Systems* 1, 1986.

Belkin, N. J., Brooks, H. M. and Daniels, P. J., "Knowledge Elicitation Using Discourse Analysis," *Knowlege Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1986.

Boose, J. H., "Personal Construct Theory and the Transfer of the Human Expertise," *Proc. of the National Conference on Artificial Intelligence*, Austin, Texas 1984.

Boose, J. H., "A Knowledge Acquisition Program for Expert Systems based on Personal Construct Psychology," *International Journal of Man-Machine Studies*, 23, 1985.

Boose, J. H., *Expertise Transfer for Expert System Design*, New York, Elsevier, 1986.

Boose, J. H. and Bradshaw, J. M., "Expertise Transfer and Complex Problems Using AQUINAS as a Knowledge Acquisition Workbench for Expert Systems," *Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada , 1986.

Buchanan, B. G., "New Research in Expert Systems" in *Machine Intelligence 10* (Hayes, J. E., Michie, D., and Pao, Y. H. eds.), Edinburgh : Edinburgh University Press, 1982.

Buchanan, B. G. and Mitchell, T.M., "Model-directed Learning of Production Rules" in *Pattern-directed Inference Systems* (Waterman, D. A. and Hayes-Roth, F. eds.), New York : Academic Press, 1978.

Bunely, A., Silver, B. and Plummer, D., "An Analytical Comparison of Some Rule Learning Programs," *Artificial Intelligence*, 27, 1985.

Buntine, W., "Induction of Horn Clauses : Methods and the Plausible Generalization Algorithm," *Knowledge Acquisition for Knowledge-Based Systems workshop*, Banff, Canada, 1986.

Burstein, M. H., "Concept Formation by Incremental Analogical Reasoning and Debugging," *Machine Learning : An Artificial Intelligence Approach 2* (Michalski, R. S., Carbonell, J. G. and Michalski R. S. eds.), California : Tioga, 1983.

Carbonell, J. G., "Learning by Analogy : Formulating and Generalizing Plans from Past Experience" in *Machine Learning : An Artifical Intelligence Approach 1 (*Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. eds.), California : Tioga, 1983.

Carbonell, J. G., Michalski, R. S., and Mitchell, T. M., "An Overview of Machine Learning," in *Machine Learning : An Artificial Intelligence Approach 1* (Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. eds.), California : Tioga, 1983.

Chandrasekaran, B., "Towards a Taxanomy of Problem Solving Types," *AI Magazine*, 4, 1, 1983.

Chandrasekaran, B., "Expert Systems : Matching Techniques to Tasks" in *Artificial Intelligence Applications for Business*, Norwood, New Jersey : Albex 1984.

Chandrasekaran, B., "Generic Tasks in Knowledge-Based Reasoning : High-Level Building Blocks for Expert System Design," *IEEE Expert* , 1, 3, 1986.

Chang, C. L. and Lee, R. C. T., *Symbolic Logic and Mechanical Theorem Proving*, New York : Academic Press, 1973.
Chomsky, N., "Three Models for the description of languages," *IEEE. Trans. Inf. Theory IT-2*, 1956.

Chomsky, N., "On Certain Formal Properties of Grammars," *Inf. Control* , 2, 1959.

Chomsky, N., "A Note on Phrase-Structure Grammars," *Inf. Control* , 2, 1959.

Cohen, P. R., and Feigenbaum, E. A. (eds.), *The Handbook of Artificial Intelligence, Vol. 3*, California : Kaufmann, 1982.

Dietterich, T.G., London, R., Clarkson, K. and Dromey, R., "Learning and Inductive Inference" in *The Handbook of Artificial Intelligence* (Feigenbaum E. eds.), Morgan Kaufmann, Los Altos, 1982.

Dietterich, T. G. and Michalski, R. S., "A Comparative Review of Selected Methods for Learning from Examples," in *Machine Learning : An Artificial Intelligence Approach* (Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. eds.), Tioga, 1983.

Duffy, F., "Dealing Rooms," *The Banker*, September 1986.

Earley, J., "An Efficient Context-Free Parsing Algorithm," *Comm. ACM*, 13, 1970.

Ericsson, K. A. and Simon, H. A., *Protocol Analysis : Verbal Reports as Data*, Cambridge, MA : MIT Press, 1984.

Forgy, C. L., *The OPS5 User's Manual*, Carnegie Mellon University, Department of Computer Science, 1981.

Gammack, J. G. and Young, R. M., "Psychological Techniques for Eliciting Expert Knowledge," *Proceeding of the 4th Technical Conference of the B.C. Specialist Group on Expert Systems*, University of Warwick, 1984.

Gentner, D., "The Structure of Analogical Models in Science," Technical Report No. 4451, Bolt Beranek and Newman, Cambridge, Mass., 1980.

Gentner,D., "Structure-Mapping : A Theoretical Framework for Analogy," *Cognitive Science*, 7, 2, 1983.

Gips, J., "A Syntactic-Directed Program that Performs a Three-Dimensional Perceptual Task," *Pattern Recognition*, 6, 1974.

Hart, A., "The role of Induction in Knowledge Elicitation," *Expert Systems* 2, 1985.

Hayes-Roth, F. and McDermott, J., "Patterns of Induction and Associated Knowledge Acquisition Algorithms," *Pattern Recognition and Artificial Intelligence* (Chen, C. ed.), New York : Academic Press.

Hendrix, G., "Discourse Analysis" in *Understanding Spoken Knowledge* (Walker, D. E. ed.), New York : Elsevier-North Holland, 1979.

Holsapple, W. C., Tam, K. Y., and Whinston, A. B., " An Induction Approach to Acquire Trading Rules," Working Paper, Krannert Graduate School of Management, Purdue University, 1987.

Johnson, S. C. ,"Hierarchical Clustering Schemes," Psychometrika, 32, 1967.

Kaufman, P., Commodity Trading Systems and Methods, N. Y., 1978.

Kelly, G. A., *The Psychology of Personal Constructs*, New York : Norton, 1955.

Kruskal, J. B., "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis," *Psychmetrika*, 29, 1964.

Lafrance, M., "The Knowledge Acquisition Grid : A Method for Training Knowledge Engineers," *Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1986.

Larson, J. and Michalski, R. S., "Inductive Inference of VL Decision Rules," *Proceedings of the Workshop on Pattern Directed Inference Systems, SIGART, Newsletter* 63, 1977.

Levenshtein, V. I., "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Sov. Phys. Dokl.* 10, 8, 1966.

MaCarthy, J., "Programs with Common Sense," in *Semantic Information Processing* (Minsky, M. ed.), Cambridge, Mass. : MIT Press, 1968.

Michalski, R. S. and Larson, J. B., "Selection of most representative training examples and incremental generation of VL1 hypotheses : The underlying methodology and the description of programs ESEL and AQ11," Rep. No. 867, Computer Science Department, University of Illinois, Urbana, 1978.

Michalski, R. S., "Pattern Recognition as Rule-guided Inductive Inference," *IEEE .Tran. on Pattern Analysis and Machine Intelligenece*, 2, 4, 1980

Michalski, R.S. and Chilausky, R. L., "Knowledge acquisition by encoding expert system rules versus computer induction from example - A case study involving soybean pathology," *International Journal of Man-Machine Studies* 12, 1980.

Michalski, R. S., "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, 20, 1983.

Michalski, R. S., "Learning Strategies and Automated Knowledge Acquisition : An Overview" in *Knowledge Based Learning Systems* (Bolc, L. ed.), Springer-Verlag, 1985.

Michalski, R. S. and Chilausky, R. L., "Learning by being told and learning by examples : an experimental comparison of two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis," *International Journal of Policy Analysis and Information Systems*, 4, 2, 1980.

Minsky, M., "A Framework for Representing Knowledge," in *The Psychology of Computer Vision* (Winston, P. ed.), New York : McGraw-Hill, 1975.

Mitchell, T. M., "Version Spaces : A Candidate Elimination Approach to Rule Learning," *Proc. of the 5th. Int. Conf. on Artificial Intelligence*, Cambridge, Mass., 1977.

Mitchell, T. M., "Generalization as Search," *Artificial Intelligence*, 18, 1982.

Moayer B. and Fu, K. S., "A Tree System Approach for Fingerprint Pattern Recognition," *IEEE Trans. Comp. C-25*, 1976.

Newell, A. and Simon, H. A., *Human Problem Solving*, New Jersey : Prentice-Hall Inc, 1972.

Quillian, R., "Semantic Memory," in *Semantic Information Processing* (Minsky, M.ed.), Cambridge, Mass. : MIT Press, 1968.

Raphael, B., "A Computer Program for Semantic Information Retrieval," in *Semantic Information Processing* (Minsky, M. ed.), Cambridge, Mass. : MIT Press, 1968.

Reid, I., "Artificial Intelligence in the Market," *The Banker*, 1986.

Rosenfeld, A., *Picture Languages*, New York : Academic Press, 1979.

Stallings, W. W., "Chinese Character Recognition," in *Syntactic Pattern Recognition Applications* (Fu, K. S. ed.), New York : Springer-Verlag, 1979.

Sammut, C. A., "Learning Concepts by Performing Experiments," Ph.D. dissertation, Department of Computer Science, University of New South Wales, Sydney, Australia, 1981.

Schank, R. C., *Conceptual Information Processing*, Amsterdam : North-Holland, 1975.

Shortliffe, E. H., *Computer-based Medical Consultations : MYCIN*, New York : Elsevier, 1976.

Simon, H. A., "Why should Machines Learn ?" in *Machine Learning : An Artificial Intelligence Approach 1* (Michalski, R. S., Carbonell, J. G. and Mitchell, T. M. eds.), California : Tioga, 1983.

Solomonoff, R. J., "A Formal Theory of Inductive Inference," *Inf. Control*, 7, 1-22, 1964.

Tanaka, E. and Fu, K. S., "Error-Correcting Parsers for Formal Languages," *IEEE Trans. Comp. C-27*, 1978.

Tversky, A., "Features of Similarity," *Psychological Review*, 84, 1977.
Waterman, D. A. and Newell, A., "Protocol Analysis as a Task for Artificial Intelligence", *Artificial Intelligence*, 2, 1971.

Winston, P. H., "Learning Structural Descriptions from Examples", in *Psychology of Computer Vision* (Winston, P. eds), McGraw-Hill, 1975.

Winston, P. H., "Learning and Reasoning by Analogy," *Communication of ACM*, 23, 12, 1980.

You, K. C., Fu, K. S., "A Syntactic Approach to Shape Recognition using Attributed Grammars," *IEEE. Trans. Syst. Man Cybern., SMC-9(6)*, 1979.

VITA

VITA

Kar Yan Tam was born in Hong Kong on April 2, 1962. He received his B.S. in Mathematics and Computer Science from University of Illinois (Urbana-Champaign) in May 1984 and his M.S. in Computer Sciences from Purdue University in May 1987. His research interests are security trading automation, expert systems and machine learning.