October 1985

Report No. STAN-CS-86-1116
Also numbered KSL-85-42

Inductive Knowledge Acquisition for Rule-Based Expert Systems

by

Li-Min Fu and Bruce G. Buchanan

Department of Computer Science

Stanford University
Stanford, CA 94305

Knowledge Systems Laboratory
Report No. KSL-85-42

October 1985

INDUCTIVE KNOWLEDGE ACQUISITION FOR RULE-BASED EXPERT SYSTEMS

Li-Min Fu and Bruce G. Buchanan

Department of Computer Science
Stanford University
Stanford, CA 94305

Table of Contents

INDUCTIVE KNOWLEDGE ACQUISITION FOR RULE-BASED EXPERT SYSTEMS

Abstract

The RL program was developed to construct knowledge bases automatically in rule-based expert systems, primarily in MYCIN-like evidence-gathering systems where there is uncertainty about data as well as the strength of inference, and where rules are chained together or combined to infer complex hypotheses. This program comprises three subprograms: (1) a program that learns confirming rules, which employs a heuristic search commencing with the most general hypothesis; (2) a subprogram that learns rules containing intermediate concepts, which exploits the old partial knowledge or defines new intermediate concepts, based on heuristics; (3) a program that learns disconfirming rules, which is based on the expert's heuristics to formulate disconf irming rules. RL's validity has been demonstrated with a performance program that diagnoses the causes of jaundice.

1 Introduction

An inductive learning program, named *RL* (for "rule learning"), has been developed for constructing and maintaining knowiedge bases (KB) in expert systems. This program differs from other inductive concept learning programs in that it can define useful new concepts which are not in the initial vocabulary in order to fill in possible missing links in a complex reasoning network. In addition, RL possesses two other distinct features. First, each learned rule is assigned a number representing "degree of certainty" (see Appendix A); second, disconfirming rules are learned as well. The learning method employed in this program is described in general terms, illustrated with its application to the problem of learning rules that correctly diagnose causes of jaundice. The performance program which uses these rules for diagnosis is called JAUNDICE.'

The problems discussed in this paper, which RL addresses, are:

- **Given: A case** library. **Find:** A set of rules that can correctly diagnose these and similar cases. (See Section 2.)
- **Given: A** faulty conclusion from the performance program. **Find:** Improvement to the KB in order to achieve a correct conclusion. (See Section 3.)
- **Given:** A case library. **Find:** Rules that predict exclusion from a class (disconfirming rules). (See Section 4.)
- **Given:** A case library. **Find:** Rules that chain together using intermediate concepts. (See Section 5.)

In the case library, each case (training instance) is represented by a frame with two basic slots: a feature set (i.e., a set of feature-value pairs), and a correct classification. The goal of the first problem is to construct a KB which can diagnose all (or most) cases correctly; the second problem is to update the KB when a new case with a faulty diagnosis appears.

In order to solve the first problem, a method which performs a heuristic search from the most general hypothesis is developed; this method, described in Section 2, is a model-driven method that processes all data at once. In order to solve the second problem of updating the KB, a "focusing mode" of learning (described in Section 2.4) is developed. When the performance program makes a faulty conclusion, rules which can achieve a correct conclusion are first learned with this focusing mode. The KB is then edited by comparing the learned rules with the old rules.

As in Meta-DENDRAL [Buchanan 78a], a half-order theory of the domain guides the learning program by providing some semantic, as well as syntactic constraints on the learned rules. In RL, this rule-forming knowledge is encoded as rule models and function templates [Davis and

'JAUNDICE is modeled after MYCIN, but was constructed from LISP. rather than from within EMYCIN, in order to allow changing parts of the program as needed for the development of RL.

Buchanan 77]. For example, some knowledge in TEIRESIAS-style function templates[2] allows association of predicates such as "SAME" and "NOTSAME" with attribute-value pairs. Semantic information, e.g., about incompatible features, is not used in the present program. The knowledge they provide is essential to make the machine-learned rules more uniformly encoded and more meaningful. The advantages brought about by this rule-forming knowledge are four-fold:

- It indicates the required components to add such that rules learned will conform to the rules written by human experts. For example, a rule learned is:

  ```
  "if P, then C"
  ```

  but a rule written by human experts may contain a screening clause:

  ```
  "if C is unknown, and P, then C".
  ```

- It tells about the commonly used predicates for a certain attribute.
- It guides generating code for newly learned rules in compliance with the old coding in the KB. Therefore, the new rules can be used by the inference engine once they are learned and can thereby be evaluated immediately to see their effects on the system performance (upgrading or degrading).
- It provides other common sense; e.g., mutual exclusivity and relative priorities among attributes. In the LHS of a rule, it allows no two conjuncts that are mutually exclusive of each other; also, the conjuncts should be ordered according to priority. If the failure of one conjunct will make other conjuncts meaningless, then this conjunct should be placed first. In JAUNDICE, for instance, it is logical to place the conjunct "LFT is known" before "SGOT is elevated*' and "SGPT is elevated" because if "LFT is not known", then no data are available for "SGOT" and "SGPT".

Rule-forming knowledge is synthesized from rules already in the KB, but can actually be formulated directly by experts as the initial knowledge for the learning system to construct a KB from nothing.

Section 3 describes learning intermediate knowledge, which is important for constructing a KB with good accuracy and understandability. Section 4 describes learning disconfirming rules. Section 5 describes how to combine all the methods developed to construct a KB with both confirming and disconfirming knowledge, including intermediate knowledge. Finally, we describe results in which learned rules are tested in the context of diagnosing the causes of jaundice.

1.1 The Case Library Used to Learn Rules for JAUNDICE

Though the methods we developed are general, we use the medical domain of jaundice as the main experimental domain. There are 72 cases in the initial case library; a feature base with 81 medical features (some binary, others with multiple values) is used for describing cases; every case in the case library has been assigned a correct diagnosis. Descriptions of cases, concepts, or hypotheses in the search space are represented by ***feature*** sets, sets of feature and value pairs: e.g., {.......(SGOT 250) (SGPT 200) (Alk-P 25).....}. The temporal characteristics are also encoded in to features; e.g., (disease-course rapid-down hill).

[2]This is currently embedded in Lisp code.

1.2 Overview of the Rule Space for JAUNDICE

The number of rules in the rule space is about 10^{25}. Thus the space is much too large to search exhaustively.[3] The space is searched systematically by stepwise generation of successors using the specialization rules Sl-S3. The rules of generalization are applied later for refinement. Section 2 discusses the procedure and heuristics for pruning the search space. The rules of generalization used are listed as follows:[4] (notation "G>" means "replaced by a more general form")

G1. Dropping conditions:

```
{(Ai Vi) (Aj Vj)}      G>      {(Ai Vi)}
```

Based on this rule, generalization is done by removing some feature-value pairs from the feature set.

G2. Climbing up the value hierarchy tree:

```
If Vi implies Vk, and Vj implies Vk,
then,
    {(Ai Vi)}          G>        {(Ai Vk)}
    {(Ai Vj)}
```

This rule states that a value can be replaced by a more general value in order to cover more instances in the same class.

G3. Taking minimum or maximum:

```
a.   {(Ai a)}          G>          {(Ai >min(a b))} ,   or
b.   {(Ai b)}                      {(Ai <max(a b))}
```

This rule is designed for numerical parameters and can be understood by the following example. For two patients with the disease '*hepatitis", if one patient's data includes "SGOT = 200", and the other's includes show "SGOT = 400", then it may be hypothesized that "SGOT $\geq$ 200" implies "hepatitis". Depending on the distribution of values among the normal and diseased populations, "taking minimum" or "taking maximum" rule is chosen. In the application of RL to jaundice, this rule is made domain-specific by adding the following heuristic: if the high range of values suggests disease, then use the "taking minimum'* rule; if the low range of values suggests disease, then use the **"taking** maximum" rule.

G4. Creating new symbols: In Section 3, we will describe how to define new symbols for higher level abstraction when indicated by some heuristics.

G5. Introducing disjunction:

```
{(Ai Vi) (Aj Vj)          G>     {(Ai Vi) (Aj Vj-or-Vk)}
{(Ai Vi) (Aj Vk) 3
```

To avoid trivial disjunction, this rule is invoked only under certain circumstances. If two case descriptions have the same values in every important feature and differ in the values of one less important feature -- and rules G1 - G4 cannot be successfully applied -- then use G5.

Rules of specialization are duals of the generalization rules. There are three rules

[3]If all 81 features were merely binary, there are 2^{81} or about 10^{24} feature sets. Each can be associated with any of about 10 disease categories. The single heuristic of looking only at rules with six or fewer features cuts the rule space to about 10^{12} rules since there are about 3 x 10^9 combinations of 81 features taken six at a time and each combination can have 2^6 possible values.

[4]Rules of generalization used in learning from examples can be seen in [Michalski 83a]. But, in JAUNDICE, we also add some domain specific rules.

corresponding to G1, G2, and G3 above (where the notation "S>" means '*specialization*' of concept descriptions or the LHS of rules),

S1. Adding conditions:

$$\{(A_i\ \ V_i)\}\ \ S>\ \ \{(A_i\ \ V_i)\ \ (A_j\ \ V_j)\}$$

S2. Climbing down the value hierarchy tree: In the tree, the highest level nodes are the most general values or descriptions: the lowest level nodes are the most specific values or descriptions. For example, the value "2" is more specific than the value "even", and the value "even" has infinite successors: . . . . -2, 0, 2, 4 ,...

S3. Closing interval: If the value is numerical and the interval is too open, then it can be specialized by closing the interval as follows.

"b" is the next higher marking level of "a".

a. $\{(Ai > a)\}$ S> $\{(Ai\ [a\ b])\}$, or
b. $\{(Ai \geq a)\}$ $\{(Ai \geq b)\}$

For example, a description ((SGOT >50)} is too general for the disease "Cholangitis" and can be specialized into ((SGOT [SO 300])} or ((SGOT ≥300)}; however the latter is not medically accurate.

The duals of G4 and G5 are not used.

The main features of RL are summarized in Table 1. Not all are discussed in the present paper. See [Fu 85] for details.

Table 1. Summary of the RL Program

Paradigm:	Learning from examples
Representation of training instances:	Feature Sets
Representation of learned concepts:	MYCIN-like rules and meta-rules with degrees of certainty
Rules of generalization:	1. Dropping conditions 2. Climbing up the value hierarchy tree 3. Taking minimum or taking maximum 4. Creating new symbols 5. Introducing disjunction
Rules of specialization:	1. Adding conditions 2. Climbing down the value hierarchy tree 3. Closing interval
Intended applications:	Expert systems in general
Validation:	JAUNDICE (REFEREE)
Efficiency enhancement:	CONDENSER Heuristics
Noise elimination:	Noise filter

Before using RL to construct a KB for JAUNDICE, we hand-coded a set of rules that gave as accurate diagnoses as we felt possible on clinical evidence alone (i.e., without laboratory tests and specialized procedures). It contains 141 diagnostic rules and an additional 80 taxonomic and causal rules. The hand-coded version of JAUNDICE was used as a reference point for performance (see Sec. 6). It was also abstracted to define the half-order theory for RL. As an alternative, the half-order theory could have been constructed without reference to the hand-coded KB -- and would be in practice where the KB is yet to be formulated. But we believed there was less bias in automatically abstracting it from a set of rules than in constructing it manually.

In order to avoid unnecessary risk and cost, the RL program is designed to keep the learned rules small (i.e., mention no unnecessary features), and keep them maximally specific while sufficiently general (i.e., the most specific rules in the version space)? Thus the procedure minimizes false positive errors (incorrect diagnoses). A subprogram, called CONDENSER [Fu 85], removes unnecessary features, thus enhancing the efficiency of learning. It also considers cost and risk associated with features, thus increasing the clinical relevance of the diagnostic rules. Another subprogram, called the noise filter [Fu 85], removes error-sources (noise) associated with learning as much as possible by optimization with respect to prediction errors. Neither CONDENSER nor the noise filter is described here. The meta-rules used for control can also be learned, but this is also described elsewhere. [Fu and Buchanan 84]

2 Learning an Initial Knowledge Base

This learning method, performing a heuristic search from the most general hypothesis, is model-driven and is designed to learn multiple disjunctive concepts (i.e., there are multiple concepts and there are multiple rules for each concept).

Given: A case library.

Find: A set of rules that can correctly diagnose cases like those in the case library.

Note that the ultimate goal is that the learned rules should diagnose cases outside the case library used for training; therefore, the learned rules should be sufficiently general and specific.

Since we use a set of training instances to estimate the "true" boundaries (i.e., rules) separating positive and negative instances, the learned rules will be associated with some degree of error. The errors concerned here are false prediction errors; i.e., the extent to which rules make wrong predictions. We particularly desire to minimize false positive predictions (in which negative instances are classified as positive) because of reasons described in Section 2.2. Therefore, this learning method is intended to discover rules that describe a group of positive training instances in a maximally specific fashion. A learned rule will tend to be overly generalized (overgeneralization implies false positive predictions) if there are no adequate negative training instances to constrain or guide generalization properly [Carbonell 83]. Hence, the strategy of finding maximally specific rules will be even more useful when few negative training instances are available.

There are domain dependent constraints for rules. The learning method searches for rules which are maximally specific without breaking the constraints. The constraints, as in the half-order theory in Meta-DENDRAL, are based on the domain knowledge. For learning in JAUNDICE, the constraints are defined as follows:

[5]Because of the involvement of uncertainty and disjunction, the term "version space" denotes the set of all plausible hypotheses with respect to a certain criterion, rather than the set of all hypotheses that are consistent with all training instances observed so far.

1. The LHS of a rule should have fewer than seven conjuncts.[6]
2. A rule should cover (be matched by) at least 20% of the positive training instances of the class for which we want to learn classification or diagnostic rules.'
3. The (absolute value of) "degree of certainty" of a rule should be at least 0.4. That is, the prediction should be reasonably certain.
4. A rule should not match more than 10% of all negative training instances.[8]

The first two constraints define minimal generality whereas the last two constraints define minimal specificity.

2.1 Procedure

In a case library, if we want to learn rules for a certain class, then label all cases in that class as positive instances and label other cases as negative instances. Inasmuch as the goal is to construct a KB covering all classes in the case library, each instance will be labelled as a positive instance of some class at a certain stage during the whole learning process.

The learning procedure includes four main steps described below. As in Meta-DENDRAL, negative instances of partial rules are not considered until the algorithm finds plausible rules based on positive evidence alone. The reason for this in both programs is the cost of matching.

- step 1. Starting from the most general version, "NIL", search for the maximally specific hypotheses (LHS's) that do not break two following constraints: the number of conjuncts should be less than seven (adjustable), and each hypothesis should cover at least 20% (adjustable) of positive instances. The hypotheses, thus found, are joined with the class name to form raw rules. Since the constraints merely involve positive instances, only positive instances are considered in this step. Since the search involves only specialization, only the specialization rules Sl-S3 are used in this step.
- step 2. Match all rules from step 1 against all cases in in the training set and prune those rules which are assigned degrees of certainty smaller than 0.4 (adjustable), or which cover more than 10% (adjustable) of negative instances. Negative instances are considered in this step for calculating degree of certainty (refer to Appendix A).
- step 3. Optimize each raw rule by iteratively applying generalization operators (generalization rules G1-G5)[9] until a local optimum is reached (through hill-climbing). The local optimum is the state with minimal prediction error under the following constraints: the local optimum should not cover more than 10% of negative instances as mentioned in step 2, and the difference of degree of certainty between the local optimum and the initial state (the raw rule) should be within 0.15. The latter constraint stems from the argument that, in EMYCIN-based systems especially, rules with small differences in CFs are adding very little to one another.

[6]This constraint takes into account the grain size of concepts used in rules. In MYCIN and in the hand-coded version of JAUNDICE. rules usually have fewer than seven components in the LHS.

'This is domain dependent. If there should be only one rule that covers all the positive instances, then the rate of coverage should be 100% ideally. But we assume there are multiple disjunctive concepts to be learned (as in Meta-DENDRAL); so it is unlikely that a single rule will cover all instances. As another example, in diagnosing acute appendicitis, the rules should be more general to cover more positive instances because the mortality of this disease is high whereas the surgical risk is small; therefore the threshold should be set higher.

[8]Though, ideally, a rule should not match even a single negative instance. this is however not probable because of uncertainty in the descriptions and classifications of cases in the case library. It is also noted that, in EMYCIN-based systems, for instance, a case in class A, which is covered by rule B inferring class B. will still be classified correctly if rule A, which infers class A and covers this case, overrides rule B; recall the phenomenon of hypothesis competition in such systems.

[9]Only GI-G3 were used in learning rules for JAUNDICE.

- step **4.** If all positive instances are covered or the rate of uncovered positive instances is below a certain threshold or the number of iterations has reached a certain threshold, then exit. Otherwise, go to step 1 and reset the constraints in step 1 as follows:

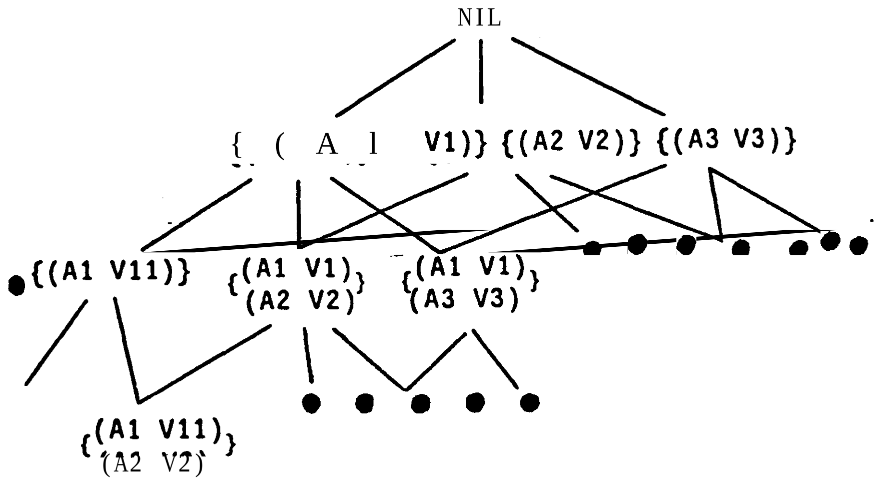
  1. Reduce the rate of coverage for positive instances; for example, the first iteration uses **20%,** the second iteration uses **10%,** and so on.
  2. The hypotheses should cover at least one of the uncovered positive instances.
  3. The number of **conjuncts** is still kept under seven.

If the constraints used in the procedure are properly chosen, only a few iterations are required, provided that the training instances are not too noisy.

The search in step 1 proceeds as follows:

- **substep** 1.1. Initialize the hypothesis space H with the most general version as follows: set H := **{** NIL **}.**
- **substep** 1.2. Generate new hypotheses by specializing each hypothesis in H in all possible **minimal** ways (i.e., specialize as little as possible). Each predecessor hypothesis may have more than one successor hypothesis. The specialization may be done by either adding a new feature with its most general value using rule **S1,** or replacing a feature value by a more specific value using S2 or S3. Figure 1 shows part of the search tree.

Figure 1. Part of the search tree.



Suppose the system is learning classification rules for "class A" instances. then the hypothesis generation in the above diagram can be formulated as:

instances => class A instances

instances with (Xl **V1)** = > class A instances

. . . . .

instances with (A1 V11) = > **class** A instances

. . . . .

instances with **(A1 V1** 1) **and** (X2 **V2)** = > class A instances

. . . . .

Note: For the feature "Al", "V11" is more specific than "V1" in the value hierarchy tree.

- Since the number of conjuncts may not exceed six, the depth of the search tree is mainly affected by (but not the same as) the depth of the value hierarchy tree, and its breadth is affected by the breadth of the value hierarchy tree and the number of the available features. Inasmuch as the search space may be huge, heuristic search is necessary. The heuristics used will be described next.

- **substep 1.3.** If a successor hypothesis is justified (i.e., does not violate the constraints defined by minimal generality, as described in step 1), then retain it in H and prune the predecessor hypothesis. If a successor hypothesis is not justified, then prune it. If no successor hypothesis is justified, then output the predecessor hypothesis as a raw new rule and remove the predecessor hypothesis from H. Also, remove redundant hypotheses during the search.

- Repeat substep 1.2 and substep 1.3 until H is empty.

If a hypothesis is pruned, it indicates that the number of conjuncts is greater than six or the positive instances covered are less than 20% of all positive instances. Successors (specializations) of this hypothesis will also be unjustified because specialization neither causes more instances to be covered nor causes the number of conjuncts to drop. Therefore, pruning unjustified hypotheses will not hurt the completeness of the search for desirable rules.

There must be a procedure to determine whether a rule is matched by an instance. An instance matches a rule when the instance makes the premise (LHS) of the rule true. Matching naturally depends on the representation language which, in our scheme, is a feature set representation.

Step 4 is designed for handling more special cases (or exceptional cases) which are not covered by rules learned in the first iteration. With more iterations, the learned rules become more specific and cover a smaller number of positive instances. The final result will be the union of results from all iterations.

The search space (i.e., the space of all possible rules) is formidable if the number of feature-value pairs is large. In our work, the search space is reduced to reasonable dimensions by:

1. Reducing the number of features used to form rules. This is accomplished by a program named CONDENSER, which is not described here. See [Fu 85].
2. Using heuristics to prune the search space.

The size of the search tree after adding heuristics may be estimated as follows:

$$B + B^2 + B^3 + \ldots + B^D,$$

where $B \leq n \times w_{max}$
$D \leq 6 \times d_{max}$

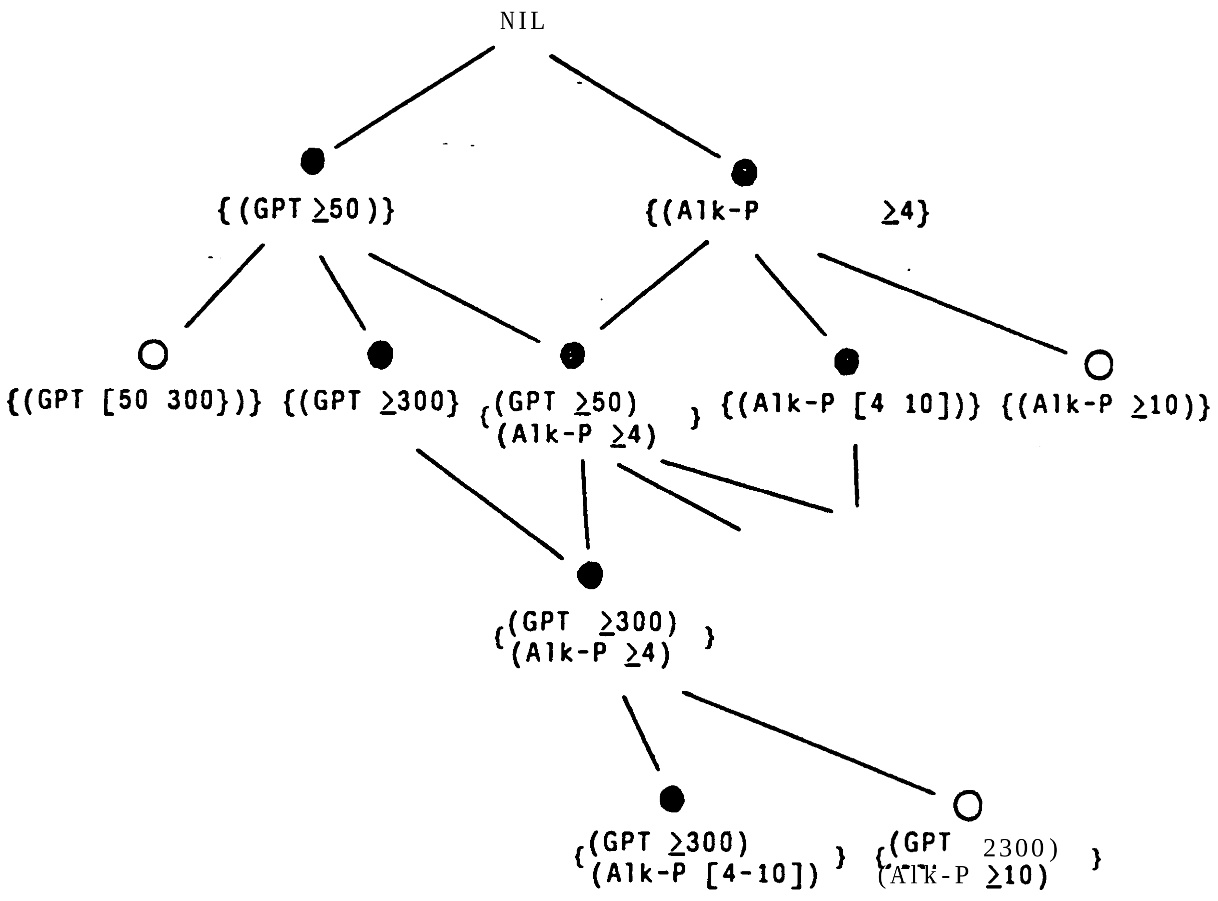
and B: Number of successors of a hypothesis.
D: Depth of the tree.
n: Total number of features involved.
w_{max}: Maximal width of value hierarchy trees among all features.
d_{max}: Maximal depth of value hierarchy trees among all features (numerical features depend on the number of intervals defined).

The number of successors of a given partial hypothesis depends on the number of features already involved in it and the number of features which can be added to it (recall the specialization process in substep 2).

Notice that the system parameters, such as "the minimal coverage of positive instances" (defined to be 20% for learning rules in JAUNDICE), can be adjusted for different domains.

2.1.1 One Example

Figure 2. The search tree of applying the "search from the most general hypothesis" method to one example.



●: Justified node
O: Unjustified node
GPT= Glutamate Pyruvate Transaminase
Alk-P= Alkaline Phosphatase

Here, an example taken from JAUNDICE is used to demonstrate the search process in the procedure described above. For simplicity, we assume only four cases in the case library, only two features in the descriptions of the cases, and only two diagnostic categories.

```
Casel: Hepatitis, {(GPT 1200) (Alk-P 8))
Case2: Hepatitis, {(GPT 450) (Alk-P 6))
Case3:.Calculous-jaundice, {(GPT 200) (Alk-P 20))
Case4: Calculous-jaundice, {(GPT 60) (Alk-P 8))
```

The value hierarchy tree rests with the domain knowledge. In JAUNDICE, the specialization of a numerical feature is done by specialization rule S3.

Now, the search tree of learning rules for "Hepatitis" is diagrammed in Figure 2. In the half-order theory, the values of "GPT" are marked off in three ranges defined by levels at 50 and 300, and the most general value is assumed to be "$\geq$50"; the values of "Alk-P" are marked off by three ranges defined by levels of 4 and 10, and the most general value[10] is assumed to be "$\geq$4". In Figure 2, the output raw rule is as follows:

```
Rl: {(GPT ≥300) (Alk-P [4 10])} => Hepatitis
```

This raw rule then goes through steps 2 and 3 described above.

2.2 Areas of Application

Inasmuch as this learning method is initially focused on systems with EMYCIN-like frameworks, its applicability is expected in domains where EMYCTN can apply, e.g., medical examples, such as MYCIN, PUFF, HEADMED, CLOT, and nonmedical examples, such as SACON [Buchanan and Shortliffe 84]. However, from the methodological viewpoint, this method can be applied in domains where there are multiple disjunctive concepts[11] and is particularly useful when:

- uncertainty is involved, or,
- false positive predictions are to be minimized, or,
- negative training instances are of limited availability.

Determinism (or exactness) of a domain will not preclude the use of the method described here since such a domain is merely an extreme case of an imprecise domain, where "degree of

[10] In the JAUNDICE experiment described in Sec. 6., the most general value for any numerical feature is "$\geq$0".

[11] A disjunctive concept in our context is a category predicted by multiple rules.

certainty" is quantized into two levels: "yes" and "no".[12] With respect to achieving accuracy of performance, false negative predictions (cases which are not predicted to be any pre-defined category) are more advantageous than false positive predictions (incorrect predictions) since for the former, a diagnostic system (or person) will continue to request desired (or missing) information which helps to attain a prediction. For example, assume there is only one rule in the KB as follows: "A1 & A2 => Class C". Then an instance in class C with A1 and no information on A2 will not be predicted to be in Class C (a false negative). However, a diagnostic system may continue to gather information about the attribute A2. In medicine, a physician might argue that, sometimes, a false negative diagnosis will be hazardous owing to delayed therapy; however, even without a (specific) diagnosis, a therapy can still be instituted immediately under the worst assumption (default therapeutic decision) while more information is being gathered for arriving at a diagnosis. However, in some cases, minimizing false negative predictions is desirable, and finding the most general rules may be necessary.

The next issue is, why and when are negative instances of limited availability? Accurate case libraries are not always readily available, and observing new cases may be time-consuming or expensive. Generating a case library from first principles or intuition may not be straightforward or may introduce unwanted bias. In medicine, for example, training instances that can be generated hypothetically by human experts are limited to more typical or simpler cases; more complex cases usually can only be obtained by clinical observation. In a relatively unexplored domain, or preference is to obtain instances is by observation.

2.3 Comparison and Discussion

2.3.1 Comparison with Related Work

For comparison, we pick up some important prototypes from the machine learning work in AI which also deal with learning multiple concepts or multiple rules. They include Meta-DENDRAL [Buchanan 78b], AQ11 [Michalski 78], and ID3 [Quinlan 83].

Meta-DENDRAL is similar to the above developed method in the following aspects:

1. both are model-driven learning systems, performing a heuristic search from the most general hypothesis;
2. both are intended to discover rules that are sufficiently general to cover new cases and sufficiently specific to avoid many false positive predictions;
3. both consider positive training instances first, and then negative training instances;
4. both use a half-order theory to constrain the search.

However, the output of the two programs may differ because of different heuristics used in the search. The RULEGEN program in Meta-DENDRAL assumes that the '*improvement ciiterion", which compares one hypothesis with its successors with respect to plausibility, increases monotonically; therefore a cleavage rule will be formed from the hypothesis space if the improvement criterion reaches a local maximum [Buchanan 78b]. In contrast, the method

[12]The system parameters in the procedure described in Section 2.1 have to be changed as follows:

B If disjunction does not occur,

- b Set the minimal coverage rate of positive training instances to be "100%" or near 100%.
- b Set the minimal degree of certainty of a rule to be "1".
- b Set the coverage rate of matching negative training instances to be "0%" or near 0%.

B If disjunction exists,

- b The minimal coverage rate of positive training instances is domain dependent
- b Set the minimal degree of certainty of a rule to be "1".
- b Set the coverage rate of matching negative training instances to be "0%" or near "0%".

described above does not assume so, and a rule will be formed only if it is maximally specific without breaking the constraints defined by minimal generality. In other words, the method seeks boundary conditions of a region bounded by the pre-defined constraints instead of seeking a local optimum. The rationale behind this is twofold:

1. unless the heuristic function used increases monotonically, the local maximum (or minimum) is not necessarily the most desirable result:
2. as described earlier, it is desirable to minimize false positive predictions. Finding the most specific rules in the version space is the most important solution if negative training instances are not easily available.

AQ11 uses the A^q algorithm [Michalski 75] and differs from the above method in the following aspects. AQ11 uses the version space approach of data-driven learning. In terms of the version space defined by [Mitchell 78], AQ11 discovers rules in the G set (the most general rules in the version space) while RL's method discovers rules in or near the S set (the most specific rules in the version space).[13] If the version space converges such that G = S, both methods will achieve the same result. There are two possible weak points for the AQ11 algorithm in EMYCIN-like frameworks:

1. If there are no adequate negative training instances to update the G set, the rules in it will be overly general and cause more false positive predictions. However, if we can ascertain that we have adequate negative training instances to guide the generalization, finding the most general rules is more advantageous in the aspect of reducing the cost of using rules because these rules tend to have a smaller number of features.
2. With the A^q algorithm, the set of rules found is incomplete. This is due to the fact that the A^q algorithm repeatedly applies the candidate elimination algorithm with a portion of positive training instances removed during each iteration, and the procedure is terminated when all positive training instances are covered by a set of rules, rather than when all desired rules are found (refer to [Michalski 75] and [Michalski 78]).

The difference between RL's method and the ID3 algorithm is derived from different representation schemes. The ID3 algorithm uses decision trees instead of production rules to represent knowledge. The weakness of the ID3 algorithm in EMYCIN-like frameworks includes the following aspects:

1. The decision tree representation is more restrictive than the production rule representation. For example, if we transform the decision tree, which is constructed by the algorithm, into a set of rules, then rules will rigidly share at least one common feature that occupies the first decision node. The distinction would be less, however, if the algorithm were intended to discover a set of decision trees.
2. Search is incomplete because, to construct the desired decision tree, features are selected on the basis of their discriminating ability with respect to some criterion. However, note that conjunction of two trivial features may be significant.
3. ID3 will fail if uncertainty is involved; i.e., some positive instances and negative instances share an identical set of feature-values.

3 Improving an Existing Knowledge Base

In order to improve a KB on the basis of experience, a different mode of learning, called the "focusing mode," has been developed. It focuses on a single instance -- usually an important false positive or false negative -- to guide the learning system to improvements in the KB. The task is formulated as follows:

[13]Note that if we say rule A is more general than rule B, it means every time rule B succeeds, rule A also succeeds, but not vice versa; the generality has little to do with the degree of certainty associated with rules.

Given: 1. A misdiagnosed case* by the performance system.
2. A case library.

Find: Rules which can correctly diagnose the misdiagnosed case, along with most of the other instances in the case library.

● : Misdiagnosis may include failure to make any diagnosis.

There are two solutions for this problem: finding confirming rules to support the correct diagnosis (or the expert's diagnosis), and finding disconfirming rules to reject the system's misdiagnoses. The former is described here; the latter is described in Section 5. Notice, however, the rules found should be ***relevant,*** i.e., they should satisfy (be matched with) the specified case.

3.0.1 Procedures

The focusing mode of learning uses the method described in Section 2.1 except for the one additional constraint that the rules should satisfy (be matched with) the specified instance.

First, label the misdiagnosed case as a "positive instance" associated with the correct diagnostic category. Then classify the case library into positive and negative instances, based on whether **or** not they have been assigned to the same category as the specified instance. For example, If a case, **Case01,** is misdiagnosed by the system as disease B whereas it should be disease A, then all cases diagnosed as disease A in the given case library are **labelled** as "positive instance" along with the misdiagnosed case; and all other cases are **labelled as** "negative instance". The learning task is to learn LHS's of confirming rules whose RHS is "disease **A",** being certain that Case01 is covered by the new rules.

The procedure includes the same four main steps, as described in Section 2.1, with minor changes noted in italics:

step 1. Starting from the most general version, "NIL,,, search for the maximally specific hypotheses (LHS's) that do not break three following constraints: ***the hypotheses should satisfy (be matched with) the specified instance,*** the number of **conjuncts** should be less than seven, and each hypothesis should cover at least 20% of positive instances. The hypotheses, thus found, are joined with the class name to form raw rules. Since the constraints merely involve positive instances, only positive instances are considered in this step.

step 2. Match all rules from step 1 against all cases in the training set and prune those rules which are assigned degrees of certainty smaller than 0.4, or which cover more than 10% of negative instances. Negative instances are considered in this step for calculating degree of certainty (refer to Appendix **A).**

step 3. Optimize each raw rule by iteratively applying generalization operators until a local optimum is reached (perform a hill-climbing search, so to speak). The local optimum is the state with minimal prediction error under the following constraints: the local optimum should not cover more than 10% of negative instances as mentioned in step 2, and the difference of degree of certainty between the local optimum and the initial state (the raw rule) should be within 0.15. The latter constraint stems from the argument that, in EMYCIN-based systems especially, rules with small differences in **CFs** are adding very little to one another.

• step 4. If there are rules learned or the number of iterations has reached a certain threshold, go to exit. Otherwise, go to step 1 and reset the constraints in step 1 as follows:

1. ***The hypotheses should satisfy (be matched with) the specified instance.***
2. Reduce the rate of coverage for positive instances; for example, the first iteration uses 20%. the second iteration uses **10%,** and so on.

3. The number of conjuncts is still kept under seven:'

The detailed search procedure is the same as that described in Section 2.1.

There are two outcomes of this learning: success and failure. If there are rules learned, then one might ask why the learning system did not find them when the KB is initially constructed by using the batch-processing learning (described in Section 2.1). In the first place, the specified case may be an exceptional one and the rules that are consistent with it can cover only a small number of other positive instances; thus those rules dealing with this exception may not be found during the initial KB construction. Secondly, the initial training set may not be a representative sample of cases in the category of interest.

On the contrary, if there are no rules learned, what does this mean? Since the objective of this learning is to find rules that are consistent with the specified instance and most of other instances, when this procedure finds no rules there must be an inconsistency between the specified instance and the others, or there must be missing data. This could happen, for example, if the expert provided the wrong diagnosis.

Incompatibility might occur between the old rules in the knowledge base and the rules found with the focusing mode. For instance, some old rule is more general, or more specific, or conflicting with some new rule. In this case, the new rules may be tried to replace the incompatible old rules to see whether the performance will be improved. But convergence is not guaranteed (see [Fu 85] for more detailed discussion).

4 Learning Disconf irming Rules

Disconfirming rules are rules which deny some facts. They can be traced back to MYCIN [Buchanan and Shortliffe 84], in which rules with negative CFs are called disconfirming rules in contrast to confirming rules with positive CFs. In our scheme, we use a degree of certainty like CFs to represent uncertainty. An example of a disconfirming rule is as follows:

$$\text{"}P \overset{-0.7}{\Rightarrow} A\text{"} \quad \text{or} \quad \text{"}P \overset{0.7}{\Rightarrow} -A\text{"}$$

This rule says if "P" exists then "A" is disconfirmed with the degree of certainty 0.7.

There are two basic approaches to forming disconfirming rules, both of which have been implemented:

1. From high frequency evidence: If some piece of evidence (clinical manifestations in medicine) frequently appears in a hypothesis (clinical diagnosis), then the absence of that evidence tends to deny the mentioned hypothesis [Miller, Pople, and Meyers 82].
2. From mutual exclusiveness or incompatibility among facts: If some evidence, E, supports a hypothesis X which is mutually exclusive with another hypothesis Y, then E tends to disconfirm the hypothesis Y.

For the first approach, in an extreme case, if some evidence appears in a hypothesis under all circumstances, then the absence of that evidence definitely denies the mentioned hypothesis. It is called a pathognomonic finding in medicine. The statement may be rephrased as follows:

$$P(e/h) = 1 \quad \Leftrightarrow \quad P(-h/-e) = 1$$

But, if P(e/h) is not 1, then it is not necessary that P(e/h) = P(-h/-e); and each conditional probability depends on the distribution of the evidence over h and -h. It is dangerous to use only P(e/h) to estimate P(-h/-e) unless we know the distribution. It is noteworthy that in MYCIN, the CF, though related to probability, is nevertheless different from probability in some aspects [Buchanan and Shortliffe 84]. And, it is misleading to use probability to measure directly the degree of belief or disbelief. In clinical practice, it is often believed to be true that if a clinical manifestation frequently appears in one disease, then the absence of that manifestation tends to disconfirm that disease. As an example in JAUNDICE, SGPT is always elevated in the disease: acute hepatitis, and the absence of SGPT elevation makes acute hepatitis unlikely. Another example of a disconfirming rule formed by the first approach is as follows:

> If there is no history of gall bladder disease, then it is unlikely (-0.5) that the disease is calculous-jaundice.

This rule is derived from the observation that some history of gall bladder disease always exists if the jaundice is caused by gall stone.

The issue of overdisconfirmation can be solved by assigning a lower degree of certainty to a disconfirming rule (unless $P(e/h) = 1$). For instance, in JAUNDICE, we use a simple mapping, such as this:

> if attribute A always (corresponding to the degree of certainty in the range $0.8 \leq CF < 1$)) appears in disease X, then the absence of attribute A ***often*** (corresponding to the degree of certainty around 0.5) rules out disease X.

By so doing, confirming rules usually override disconfirming rules to make conclusions if both succeed.

The second approach may be represented as a rule:

> If e => h1 and hl => -h2, then e => -h2.

This approach involves more semantics than the first. Statements about incompatible concepts (e.g., hl => -h2) are represented in a so-called "denying tree," which is part of the half-order theory given to the program.

Sometimes, a disconfirming property can propagate along a relational chain (e.g., causal links); **thus:**

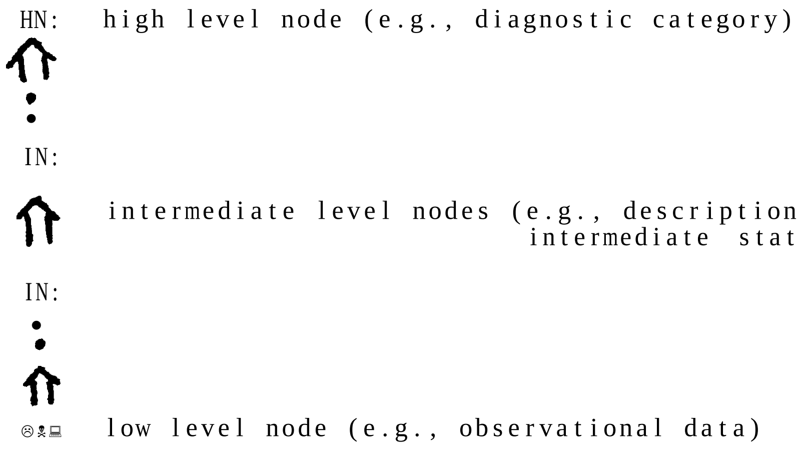
> If el => e2, e2 => -e3, and -e3 => -h, then el => -h.

The uncertainty may also propagate: the degree of certainty of a path is the product of the involved links.

Learning disconfirming rules can also focus on a specified case (focusing mode). For example, if a case is misdiagnosed as disease B while it should be diagnosed as disease A, then, with the approach from high frequency evidence, disconfirming rules can be formed to disfavor disease B by- using feature-values that are absent in the specified case but frequently present in the cases correctly diagnosed as disease B.

5 Learning Intermediate Knowledge

In expert systems, hierarchical reasoning can provide better accuracy and understandability [Clancey 83]. Descriptions of intermediate states partially summarize and categorize subsets of the data and thus allow a reasoning system to reason from initial data to final conclusions in orderly stages. In MYCIN, for example, an inferred intermediate state of the patient is “compromised host”, i.e., a person whose immune system has been lowered. It is neither a piece of primary, observed data (called here a low level node or LN) nor a final diagnostic category (called here a high level node or HN). Using intermediate nodes (INs) allows the reasoning to proceed in smaller steps. It also allows the system to exploit the familiarity of some **INs** for purposes of explanation [Clancey **83]**. Finally, **INs** may provide a partial interpretation of the data that is useful even when insufficient data are available for a complete interpretation. The reasoning hierarchy may be diagrammed as Figure 3.



HN: high level node (e.g., diagnostic category)
IN:
intermediate level nodes (e.g., descriptions of intermediate states)
IN:
low level node (e.g., observational data)

Figure 3. Multi-level reasoning network. Note that there may be several intermediate levels.

Intermediate nodes can be names of (a) generalized findings (e.g., clinical syndromes, which often start out as almost arbitrary labels for collections of findings); (b) generalized diagnostic classes (e.g., MYCIN inferred "meningitis" before it inferred "bacterial meningitis*'); (c) intermediate concepts that are neither final solutions nor primary data (e.g., "compromised host" in MYCIN).

In building a knowledge base for an expert system, it is useful to include **INs** for the three reasons just cited. If the knowledge base is built through knowledge engineering, an expert can supply the appropriate **INs.** If it is constructed automatically, however, the learning program must be able to supply the **INs** if the cases from which it learns are described in terms of **LNs** and categorized in terms of **HNs.** Thus the problem we are addressing in this paper can be described as:

Given: A set of training instances associating **LNs** and **HNs** (written LN -> HN).

Find: Rules of the form LN => IN and IN => HN consistent with the training instances.

In the above formulation, **"->"** represents a specific association in a training instance; **"=>"** represents general inferential **knowledge** (a rule). That is, we intend to learn general intermediate knowledge from a set of very specific descriptions. (Note that "LN" can be a conjunction of more than one feature.) Practically, this is an important issue and worthwhile to explore because, for instance, medical records are often described only by clinical manifestations and disease diagnoses, where discussion of the involved intermediate mechanisms is limited or missing.

Two sets of learning methods are described below to cover two important cases:

1. Intermediate nodes already exist in the initial vocabulary and are used in some rule (or in the half-order theory);
2. Intermediate nodes are **not** in the initial vocabulary.

In the first case, the learning task involves exploiting the old partial (incomplete) intermediate knowledge to learn new intermediate knowledge; the partial knowledge may exist between LN and IN, or between IN and HN, or both. The task in the second case is comparatively more abstract and constructive, because it involves creating and defining new intermediate concepts which are not provided in the original vocabulary. In inductive concept learning, automatically adding new symbols is one way to mitigate the bias induced by a fixed language [Utgoff 82].

We focus here on learning diagnostic rules that include intermediate concepts (i.e.' LN => IN and IN => HN). We first describe the learning techniques when the intermediate concepts are already in the initial language (Section 5.1), and then describe the method when the intermediate concepts are not in the language (Section 5.2).

5.1 Intermediate Concepts Already in the Initial Vocabulary

In this section, we assume there is some knowledge about the intermediate nodes (IN) including partial but not complete knowledge about their relationship with other nodes at other levels (LN or HN). We also assume each training instance is characterized only by LN and HN and not by IN. If each training instance is characterized by intermediate level concepts (IN) as well as by low level descriptions (LN) and high level concepts (HN), we can apply machine learning algorithms level by level and discover new knowledge in different levels.

Basically, two methods are used to search the space of connections of LNs to INs and INs to HNs: bottom-up and top-down. The bottom-up method relies on existing knowledge of the form "LN => IN" and "IN => LN" (written together as "LN <=> IN"). The top-down method relies on the existing knowledge of "IN <=> HN". Therefore, if only knowledge of linkages between LNs and INs is available, only the bottom-up method can be used: likewise if only knowledge of linkages between INs and HNs is available, only the top-down method can be used. If knowledge of both types is available (but incomplete, otherwise there is nothing to be learned), both methods can be applied and the results will be the union of results from each method. A third method called "bidirectional extension" employs these two basic methods bidirectionally and sequentially in order to construct more complex hierarchical concepts.

5.1.1 Bottom-Up Learning

If knowledge of the form "LN => IN" is available, the bottom-up method is applied. The task of learning under this situation is described as follows:

Given: 1. A set of training instances
(or a set of LN -> HN);
2. Associations from the
half-order theory linking LN and IN,
in the direction of LN => IN.

Find: Rules of the form IN => HN,
consistent with the training instances.

Each training instance is represented as a set of LNs and is classified on the basis of some HN. Remember that HN is a class name (or a category in medicine). The basic idea behind this method is to generalize from instances of the same HN. In the JAUNDICE experiments, rules in the form of "LN => HN" are first learned by the method described in Section 2 from the given set of training instances; then we treat this set of rules as another set of training instances (more general than the original training set, of course) and apply the following procedure to learn intermediate rules.

Suppose there are n HNs: H1, H2, . ..Hi. . ..Hn, in the set of final hypotheses. The algorithm proceeds as follows:

For i=1 to n, Do:

- step 1. Label all instances associated with the class of Hi as positive instances and label other instances as negative instances.
- step 2. Generalize from positive instances by using the concept hierarchy- tree (in the half -order theory). The generalization should:
  - o be maximally specific to avoid over-generalization;
  - o be tested against negative instances.

  Intermediate nodes are involved and intermediate links (IN => HN) are discovered during the process of generalization via hierarchy (see the following example).

Step 2 proceeds as follows:

- substep 2.1. Initialize the hypothesis space H with the set of all positive instances; i.e., each positive instance represents one hypothesis in H. For example, if hl is the first instance of class Hi, we formulate one hypothesis in H as follows: hl => Hi. Each hypothesis in H is associated with a degree of certainty, which is estimated from the statistics among instances.
- substep 2.2. For each hypothesis hi in H (starting from the head of H), do the following:
  - o Form set H* by finding all hypotheses in H with the same range of degree of certainty ($\pm$0.15) as hi (H' excludes hi).
  - o For each element hj in H*, find the common maximally specific generalization (called hk) of hi and hj. hk is plausible if it does not break the following constraints: its associated degree of certainty is at least 0.4 and in the same range as hi's and it should not cover more than 10% of all negative instances.[14] If it is plausible, then retain hk, put it in the end of H, and prune hi from H. If no element in H* can form a plausible generalization (without breaking the constraints above) with hi or H* is an empty set, then output hi as a new rule and prune it from H.
- substep 2.3. Remove redundant hypotheses from H.
- substep 2.4. Repeat substeps 2.2 and 2.3 until H is empty.

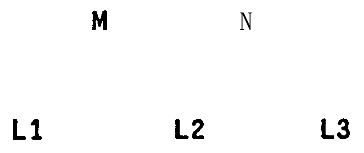
This data-driven algorithm differs from the version space approach [Mitchell 78] in that this algorithm considers not only that there may be multiple rules for each concept but also that an instance may be covered by several rules instead of a single rule. This technique is extended from the technique of climbing the generalization tree, which is used in other work, such as [Winston 70] and [Michalski 83b]. However, we emphasize a ***concept*** hierarchy instead of a ***value*** hierarchy. Moreover, uncertainty may be involved in the hierarchy.

In the following simplified example, we assume that no uncertainty is involved and that two hypotheses are mutually exclusive. (Both assumptions can be removed.)

[14]We do not consider the minimal generality here because, as stated earlier, this method is applied to the set of rules learned; therefore they are already sufficiently general.

Example 1. Suppose there are three instances (Xl, X2, X3)
in the instance space as follows:
Xl: L1 -> H1
X2: L2 -> H1
X3: L3 -> H2

A simple hierarchy is given by five rules,
given two INs (M and N) in the vocabulary
and shown schematically as follows:



The following are two possible generalizations from Xl and X2:

G1: M => H1
G2: N => H1

G2 is not justified because if it is true,
then L3 => N => H1, contradicting X3.
Thus, the new rule is: G1: M => H1.

In JAUNDICE, for example, the two rules, “esophageal varices => hepatic cirrhosis” and "ascites => hepatic cirrhosis”, may be generalized into “portal hypertension => hepatic cirrhosis” by using the existing knowledge, “esophageal varices => portal hypertension” and "ascites => portal hypertension”.

51.2 Top-Down Learning

If knowledge exists linking INs and HNs, the technique of top-down learning can be applied. The task is formulated as follows:

Given: 1. A set of training instances.
(or a set of LN -> HN):
2. Associations from the
half-order theory linking IN and HN,
in the direction of HN => IN.

Find: Rules of the form LN => IN,
consistent with the training instances.

In order to learn rules of the form "LN => IN”, we may first, based on the available knowledge of “HN => IN”, re-label training instances (rules) such that they have new class names which are IN instead of HN. The algorithm used in learning “LN => HN" is then applied to the transformed instances, and the results will be of the form "LN => IN” which is consistent with the training instances.

In JAUNDICE, for example, three diseases “acute hepatitis”, “chronic hepatitis”, and “hepatic cirrhosis”, can be transformed into a common intermediate pathological category, “hepatocellular injury”, and the inference rules for “hepatocellular injury” are learned from the same case library by the same learning method we use to learn the inference rules for each disease.

51.3 Bidirectional Extension

This strategy combines the bottom-up and top-down methods to learn more complex hierarchical concepts. Consider a four-level hierarchy as follows:

$LN \Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow HN$

Suppose we have the knowledge of "$LN \Leftrightarrow IN_{level1}$" and "$IN_{level2} \Leftrightarrow HN$" (see Figure 4) and each training instance is described by "$LN \rightarrow HN$".

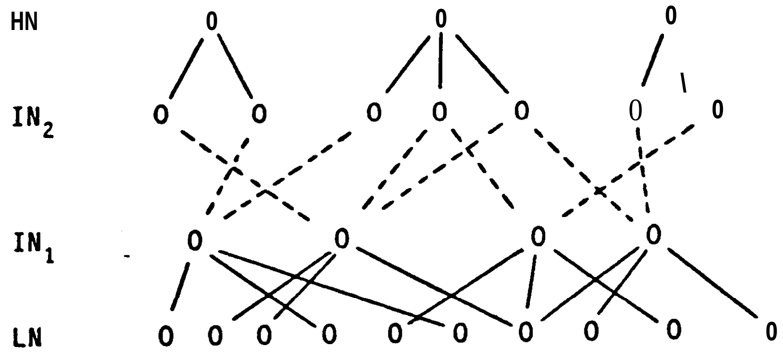


Figure 4. Diagram of the bidirectional extension strategy. Solid links represent existing knowledge; dotted links represent knowledge to be explored.

We can first learn "$IN_{level1} \Rightarrow HN$" by the bottom-up method which exploits the existing knowledge of "$LN \Leftrightarrow IN_{level1}$". Then we treat all links of "$IN_{level1} \Rightarrow HN$" as another set of training instances, and we can learn rules of the form "$IN_{level1} \Rightarrow IN_{level2}$" by the top-down method which exploits existing knowledge of "$IN_{level2} \Leftrightarrow HN$". Thus, we obtain all inferential knowledge "$LN \Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow HN$" from a set of training instances by extending the knowledge of only "$LN \Leftrightarrow IN_{level1}$" and "$IN_{level2} \Leftrightarrow HN$". . . If we view this learning task as a search, it actually alternates between bottom-up and top-down search. Whether the procedure starts bottom-up or top-down does not matter if we assume the training instances are correct and complete. In a domain with uncertainty, consider when inconsistency occurs in the following three instances (H1 and H2 are mutually exclusive): (L1 & H1), (L1 & H2), and (L2 & H1), and assume we have existing knowledge as follows: L1 => I1, L2=> I1, H1 => I11, and H2 => I12; then starting with the top-down method, we can first find the following consistent intermediate knowledge: L2 => I11, whereas starting with the bottom-up method, we find no consistent intermediate knowledge. In the current implementation, we ignore such inconsistency.

In the example of a four-level hierarchy, we can still obtain knowledge of all levels by using the bottom-up method alone if only the knowledge of "$LN \Leftrightarrow IN_{level1}$" and "$IN_{level1} \Leftrightarrow IN_{level2}$" is available, or by using the top-down method alone if only the knowledge of "$IN_{level2} \Leftrightarrow HN$" and "$IN_{level1} \Leftrightarrow IN_{level2}$" is available. Hence, it is possible to obtain even more complex hierarchical concepts by applying these two methods sequentially, depending on the available knowledge.

One example cited from JAUNDICE is as follows. Initially, there are three rules in the form of "$LN \Rightarrow HN$": "malaise => hepatitis**, "fatigue => hepatitis", and "poor appetite => hepatitis". These rules are generalized to the form of "$IN_{level1} \Rightarrow HN$" by using the bottom-up method: "constitutional symptoms => hepatitis". This rule subsequently results in another rule in the form of "$IN_{level1} \Rightarrow IN_{level2}$" by using the top-down method: "constitutional symptoms => hepatocellular injury".

5.2 Intermediate Concepts not in the Initial Vocabulary

Sometimes intermediate concepts are not already specified in the initial vocabulary, so it is necessary to create and define them. The key issues are when and how to create new intermediate concepts. Two techniques are introduced: naming taxonomy points and naming swi tchover points.

5.2.1 Technique of Naming Taxonomy Points

We assume there are n classes of objects or concepts in the instance space. The algorithm proceeds as follows:

- step 1: Construct a taxonomy tree on the basis of a similarity or dissimilarity measurement. One way of measuring dissimilarity is based on the sum of (the absolute value) of the differences **of** weighted individual features" (though non-linear functions can also be tried). First, based on domain knowledge, select some important features and assign them weighting factors. Second, calculate the difference between the average value of an individual feature for the given two classes. If the feature values are not numerical, transform them into numerical values on the basis of domain knowledge. In medicine, this is a feasible approach because the clinical feature values can be quantized according to their clinical severity. (However, in some domains, symbolic measurements may be necessary. One example of clustering by a non-numerical technique is seen in [Michalski 83a].) For example, in JAUNDICE, the elevation of the serum enzyme is quantized into 0, 1, 2 and 3, representing normal, mildly-elevated, moderately-elevated and highly-elevated. Third, calculate the sum of the differences of weighted individual features. Using different dissimilarity functions (i.e., using different features or different weighting factors) may yield different results. So, it is possible to build more than one taxonomy tree.

Example 2. Computation of dissimilarity between two diseases in a library of four cases. The weights of the two features used are assumed equal here for simplicity.

```
Instance1: {(GOT 2) (Alk-P 0)} -> Disease A
Instance2: {(GOT 3) (Alk-P 1)} -> Disease A
Instance3: {(GOT 1) (Alk-P 2)} -> Disease B
Instance4: {(GOT 2) (Alk-P 2)} -> Disease B

  (where 0 = normal
         1 = mildly abnormal
         2 = moderately abnormal
         3 3 severely abnormal)

Compute as follows:

  Average value for GOT:
    Disease A: 2+3/2=2.5
    Disease B: 1+2/2=1.5

  Average value for Alk-P:
    Disease A: 0+1/2=.5
    Disease B: 2+2/2=2

  Dissimilarity(A & B)=(2.5-1.5) + (2-.5)
                      =2.5
```

Then we set up a criterion to group different classes in a common category if their mutual dissimilarities are smaller than a certain threshold. The criterion should be set in a way such that one class will not be grouped in two different categories. In a taxonomy tree, the tip nodes are HN in our terminology, and each non-tip node (excluding the root node) represents an IN.

- step 2: Assign a symbol to every intermediate node in the taxonomy tree, and thus create new intermediate concepts (IN) (see Figure 5).

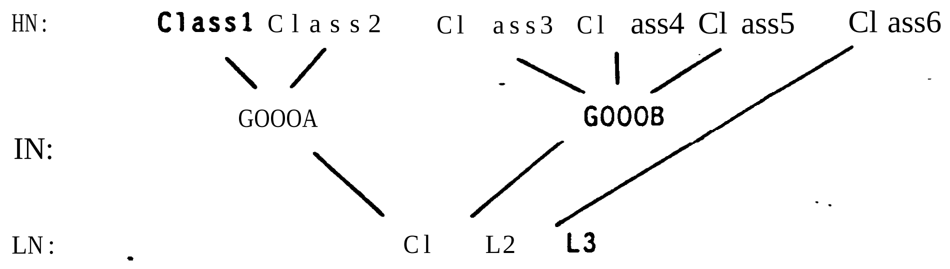


Figure 5. Suppose a simple taxonomy tree is built, the intermediate taxonomy points are named as GOOOA and GOOOB.

- step 3: Apply the top-down method (described previously to learn the knowledge of the form "LN => IN".) The taxonomy tree gives us knowledge about "HN => IN". For example, in Figure 5, "if X is a member of class 1 then X is in category GOOOA".
- step 4: Learn knowledge of the form "IN => HN" which is the link from IN (or mixed IN and LN) to HN by the following procedure:
  - First' Learn discrimination rules for different classes (HN) under the same intermediate category (i.e., under a higher taxonomical category) after removing all instances which do not belong to it. Thus, these rules are "local" rules in the sense that they are only good for a certain intermediate category. For example, in Figure 5, we may learn classification rules for class 1 and class 2 in the category GOOOA by removing all instances that are not in the category GOOOA. Suppose we obtain such a classification rule for class 1 as follows:

```
"If an object has attribute L1
  then it belongs to class 1."
```

  - Second, we actually can write a more specific rule as follows:

```
"If an object is in category GOOOA
    and has attribute L1
 then it belongs to class 1."
```

The algorithm may be applied level by level, and the results will become hierarchical; i.e., LN $\Rightarrow IN_{level1} \Rightarrow IN_{level2} \Rightarrow \ldots \Rightarrow$ HN.

In JAUNDICE, by applying this technique to 72 cases, we found five concepts (see Table 2). Four symbols, after medical interpretation, were found to correspond to "hepatocellular injury", "cholestasis", "intrahepatic jaundice", and "extrahepatic jaundice." (These terms were used in rules in the hand-coded version of JAUNDICE, but this was not known to the learning program.) A fifth term, "hemo-gilb", was found because two diseases, "hemolysis" and "congenital conjugation defect (e.g., Gilbert's disease)" are similar and under the same taxonomy point. Though clinically meaningful (corresponding to negative bilirubinuria), the concept "hemo-gilb" bears little pathophysiological meaning.

Symbols	Medical Interpretation*
Neosyml	Cholestasis
Neosym2	Extrahepatic jaundice
Neosym3	? Hemo-gil b
Neosym4	Hepatocellular injury
Neosym 5	Intrahepatic jaundice

*The interpretation depends on the diseases (HNs) and features (LNs) linked by the symbol. Neosym3 is the one term not already used in the rules and its interpretation is not as medically clear as the others.

Table 2. New symbols created by the technique of symbolizing taxonomy points.

Note that this technique is intended to discover new intermediate concepts, but the concepts may have already been in the vocabulary. Hence, after new symbols are created, they should be checked to determine whether they are semantically equivalent to the old symbols? Of course, this depends on the size of the case library, so we may want to keep redundant concepts around for a while on the assumption that future cases may further differentiate them.

5.2.2 Technique of Naming Switchover Points

The technique of naming switchover points is motivated by the observation that intermediate concepts often serve as switchover points in a reasoning network. One heuristic behind this technique is:

HR1: I f i) There are subsets of n LNs and m HNs, such that
ii) <u>All</u> n LNs have (confirming) links to <u>all</u> m HNs, and
iii) $n>1$ and $m>1$ and $mn>4$.
then it is worthwhile to define a common intermediate node.
This heuristic is also represented in Figure 6.

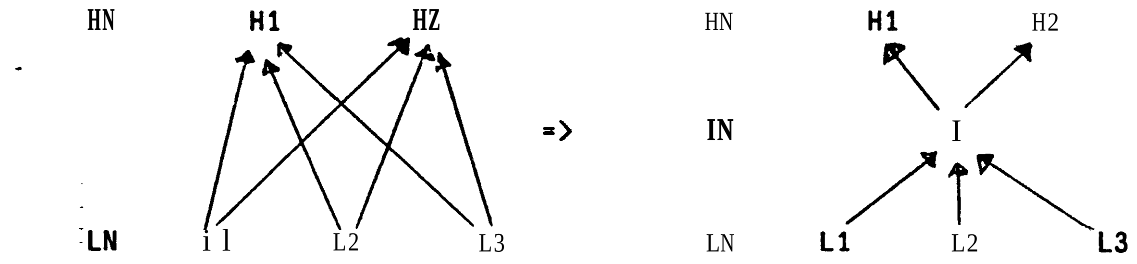


Figure 6. Creating new intermediate nodes at switchover points.

If one set of LNs (call it set L) and one set of HNs (call it set H) satisfy this rule, then many subsets of set L and any subset of set H can also satisfy this rule. Thus we determine that the intermediate node be defined on the basis of the largest sets (subsets or supersets of set L and set H) of LNs and HNs which satisfy this rule. The third condition of this heuristic is, in

[15] Checking for semantic equivalence must be done by hand at the moment.

fact, the threshold of the complexity of the relationship between LN and HN for defining new symbols. We deliberately chose this threshold because of the fact that, for a given situation which satisfies this rule, descriptions of the inference behavior are simplified by adding a common intermediate node while all links from LNs to HNs are maintained via the intermediate node (i.e., no links from LN to HN are added or removed). For example, in Figure 6, there are 6 links (LN => HN) initially and 5 links (LN => IN and IN => HN) after introducing an intermediate node. Consider a case where there are 10 LNs and 10 HNs and 10x10=100 links initially; only 20 links are needed after introducing a common intermediate node. But if $n=1$ or $m=1$ or $nm=4$ (e.g., $n=2$ and $m=2$), the descriptions of inference behavior will not be simplified by adding a common intermediate node. However, the descriptions of inference behavior may become complicated rather than simplified when so many intermediate nodes are introduced and there are overlaps of the associated sets of LNs and HNs among them (note that each newly defined intermediate node has one set of LNs and one set of HNs associated with it). Though complication is worthwhile if more understandability and better accuracy are gained, we might attempt to control the number of the newly defined intermediate nodes (concepts) by adjusting the threshold of complexity for defining them (i.e., adjusting the third condition of the described heuristic rule).

Creating a new intermediate node will face another problem if uncertainty is involved. For the example shown in Figure 6, the final degree of certainty of H1 and H2 concluded from L1, L2 and L3 should remain approximately the same before and after introduction of intermediate concepts. The degrees of certainty are assigned to new links in such a way as to preserve these final degrees of certainty?

In our experiment, heuristic rule HR1 is applied to a set of rules (LN => HN) which are learned from training instances (LN -> HN), and can be applied recursively, as long as there are plausible switchover points, to form a multi-level network. By applying this technique to the jaundice domain, nine symbols were created, which are shown in Table 3.

Symbols	Medical Interpretation*
Neosym1	Benign hepatic pathology+
Neosym2	Cholestasis
Neosym3	Chronic liver failure
Neosym4	Complete biliary obstruction+
Neosym5	Extrahepatic jaundice
Neosym6	Hepatobiliary pathology
Neosym7	Hepatocellular injury
Neosym 8	Inflammation+
Neosym9	? Liver cachexia+

*The interpretation is made by observing the involved feature (LN) and diseases (HN).

+These symbols are outside the initial vocabulary. The interpretation of Neosym9 is not as clear at that of the others.

Table 3. New symbols **created by the technique of symbolizing switchover points.**

Among these nine symbols, four symbols are outside the vocabulary of the hand-coded KB and five symbols are semantically equivalent to some old symbols. (As before, the old vocabulary was not used except for the LNs and HNs.) It is also noticed that there is some overlap of the results from the techniques of naming taxonomy points and naming switchover points. The

[16] Because there are fewer links among concepts after introducing intermediate terms, we have fewer unknown CFs on the LN => IN and IN => AN links than we had known CFs on the LN => HN links. Thus we can find approximate solutions, but cannot always satisfy all equations with precision.

fact that most of the created symbols are medically meaningful is expected because an intermediate symbol is created only when there is a complex but regular relationship between LN and HN and most of these have been named in the last 2500 years of medicine.

5.3 Related Work

Related work on creating new terms includes EURISKO [Lenat 83], BACON [Langley 83], and Utgoff's program [Utgoff 82]. The main difference is our explicit attempt to discover new intermediate concepts to construct a reasoning hierarchy. From the viewpoint of establishing a conceptual hierarchy, the most representative related work in AI is [Michalski 83a]. However, it differs from our work in two important aspects. First, our work deals not only with conceptual clustering but with finding intermediate links. Second, because each training instance is also characterized by a high level concept besides low level descriptions, the search for the meaningful intermediate concepts is constrained bidirectionally (from LN and from HN) while this is not true for [Michalski 83a].

We expect the methods described here can be easily extended to non-medical domains. In learning intermediate knowledge, we use a general concept hierarchy; and the heuristics we use to discover intermediate knowledge are not specific to medicine.

6 Results

This section describes the application of RL to constructing a hierarchical knowledge base for the domain of jaundice. We encoded a training set of 72 jaundice cases from the medical literature, using 81 features and 12 disease categories. From this case library, we constructed a knowledge base using a combination of the techniques described above.

The procedures are as follows:

- step 1. Learn the direct inference rules (LN => HN) from the given set of training instances (the method is described in Sections 2 and 4).
- step 2. Starting from partial or no intermediate knowledge, explore the intermediate knowledge by all methods that include bottom-up, top-down, bidirectional extension, naming taxonomy point, and naming switchover point, as much as possible. Two things are expected: first, some methods may not work because of incomplete knowledge, e.g., the bottom-up method cannot be adopted when knowledge of the form "LN <=> IN" is missing; second, the results from different methods may be redundant. The first problem is handled simply by abandoning the methods that can't apply. The second problem can be solved by checking and removing redundancy. The symbols created by naming taxonomy points and naming switchover points must be interpreted before checking redundancy with old symbols and new symbols already created. The interpretation can be made automatically by observing the involved LN and HN (see Tables 2 and 3). At this stage, the knowledge base under construction has the knowledge of three types: LN => IN, IN => HN, and LN => HN.
- -step 3. Replace direct rules (LN => HN) by intermediate rules (LN => IN, and IN => HN) if they are equivalent. By "equivalent", we mean the same conclusion (HN) with the same strength (degree of certainty, allowing an error of 0.15) can be reached, given a set of low-level features (LN). For instance, in JAUNDICE, a direct rule "negative bilirubinuria and elevated urobilinogen => hemolysis" can be replaced by the rule "negative bilirubinuria and elevated urobilinogen => overproduction of bilirubin" and the rule "overproduction of bilirubin => hemolysis". Note that one direct rule may be replaced by several intermediate rules.

After this procedure, the knowledge base contains hierarchical concepts, but will also contain some simple associations of the form "LN => HN" which cannot be explained by intermediate concepts.

The KB constructed automatically has 232 rules including 112 rules with intermediate concepts. We then compared this new knowledge base with an old knowledge base of 141 rules, which was built by manually encoding medical knowledge from textbooks and journals and is also hierarchically structured. First, we tested each knowledge base on the original 72 cases; the diagnostic accuracy of the new vs. the old knowledge base is 97.2% vs. 84.7%. But since the new knowledge base is based on these 72 training cases, its better performance is expected. Therefore, we further tested the knowledge base on 68 additional cases obtained from Stanford Medical Center. These cases received liver biopsy in 1978 and were ***not all*** diagnosable from clinical parameters alone. The diagnostic accuracy of the new vs. old knowledge base is 72.1% vs. 76.5%. We then removed all non-diagnosable cases among these 68 cases.[17] On the 42 remaining diagnosable cases, the diagnostic accuracy is 83.3% vs. 88.1%. The result of a paired t test shows t=1.434, which indicates that there is no significant difference between results obtained from the new KB and the old KB (see Table 4). The 5% difference in results may be ascribed to the fact that many more cases than 72 are needed to learn rules for even a well-circumscribed domain. Textbooks, after all, encode summaries of considerably more experience.

	Old KB (141 rules manually encoded from textbooks)	New KB (232 rules automatically learned)
Training set for automatic learning **(72 cases)**	**84.7%**	**97.2%**
lest set **(68** cases)	**76.5%**	**72.1%**
Test set with clinically diagnosable* cases only **(42** cases)	08.1%	**83.3%**

*Among the 68 test cases, 42 cases are diagnosable clinically (refer to text description).

Table 4. Diagnostic accuracy of automatically learned rules.

If we turn off the intermediate knowledge learner and learn only direct rules (i.e., only step 1 described above is turned on), we obtain a knowledge base of 185 (direct) rules; this knowledge base without intermediate knowledge can save execution time to some extent if compared with the knowledge base of 232 rules (since the average system execution time is roughly proportional to the number of rules in the knowledge base), but the diagnostic accuracy tested by the 42 diagnosable liver biopsy cases drops to 61.9% (vs. 83.3% if intermediate knowledge is added). Here, we may notice there is a tradeoff between execution time and quality of performance. We further notice that cases which can be diagnosed correctly by the knowledge base with intermediate knowledge and cannot be diagnosed correctly without intermediate knowledge are cases with incomplete data. It seems clear that intermediate knowledge can improve the system's predictive power particularly if only partial information is available. Moreover, intermediate knowledge provides better understandability. For instance, the

[17]By "non-diagnosable" we mean the pre-biopsy diagnosis made by the physician who sent the biopsy did not coincide with the biopsy diagnosis. Note that not every clinical case is clinically diagnosable because a disease may be in its incipient stage without full manifestation. Although some error may be introduced by using the diagnosis of the attending physician as a "gold standard", this is preferable to the bias that would have been introduced if we had selected the cases that were not diagnosable from clinical parameters alone.

incorporation of intermediate knowledge can explain 'the underlying pathological and anatomical mechanisms of jaundice and make the diagnosis more convincing. Based on the above considerations, learning intermediate knowledge is justified and desirable in expert systems.

We also tested RL in a non-medical domain as a check on its generality. The REFEREE program [Haggerty 84] is an EMYCIN system designed to critique journal articles using a KB of about 200 rules. Rules learned by RL from a case library were shown to have about the same predictive power as the hand-coded rules (see [Fu 85] for details).

7 Conclusion

We believe a model-driven learning strategy is preferable to a data-driven strategy in rule based systems for the following reasons.

- ***Completeness:*** The heuristics used in pruning the rule space in RL still allow systematic search of the rule space (see Section 2.1). Intuitively, since our goal is to discover all desirable rules, a model-driven search in the rule space (whose size is approximately the power set of the set of all descriptors used to describe rules) will tend to be more complete than a data-driven search in a subspace of the rule space.
- ***Noise immunity:*** Model-driven learning systems are more immune to noise than data-driven learning systems [Dietterich 83]. Because model-driven techniques are intended to find rules that are good in a global sense (i.e., there is a global criterion to evaluate rules), the effect of noise associated with the individual data (e.g., false positive or false negative training instances) can be mitigated under the assumption that the imperfect training instances are the minority.[18] In contrast, data-driven techniques handle the instances one at a time, and thus have more difficulty with noise associated with the data. One false positive instance will force a rule to be overly generalized while one false negative instance will force a rule to be overly specialized.
- ***Multiple fines of reasoning:*** In EMYCIN-based systems, conclusions are reached by combining several different rules that use different sets of features and develop different lines of reasoning. Several simple rules are used rather than a single long rule. For example, in JAUNDICE, the number of conjuncts in the LHS of a rule is restricted below seven. The combinatories of successively generalizing from instances, where each instance has more than seven attributes, become prohibitive -- especially when multiple lines of inference have to be maintained.
- ***Efficiency:*** In a domain with multiple disjunctive concepts, if a model-driven method is used, the search space will be roughly the power set of the set of all descriptors involved. If a data-driven method is used, the search space can be roughly estimated from the power set of the set of all positive instances since the system has to determine which group of instances should be hypothesized together. If the number of descriptors is greater than the number of positive instances, it may be more efficient to use a data-driven approach (if we ignore the disadvantages described above). Nonetheless, in real practice, the number of positive instances is often greater than the number of the descriptors. Furthermore, the CONDENSER program can reduce the number of features during learning in order to enhance efficiency [Fu 85].

The ultimate goal of this work is to develop a domain-independent program that can construct a reliable knowledge base in domains where (a) multiple disjunctive rules need to be learned, (b) case data may not be 100% reliable, (c) disconfirming rules may be of value, and (d) intermediate concepts may be of value for accuracy and understandability. The major contribution of this work is the method of learning intermediate-level concepts from a set of training instances that are described only by low level features and high level concepts and not by any intermediate concept.

[18] This assumption often holds: if not, then no inductive method can learn good rules.

The effectiveness of the RL program, as demonstrated in JAUNDICE and REFEREE, confirms the value of the model-driven learning strategy. Although our philosophy favors finding the most specific rules in the version space, the RL program can be modified to find the most general rules by generalizing rules as much as possible so long as the performance is maintained. In turn, it confirms the power of heuristic search for rule formation as also demonstrated by Meta-DENDRAL [Buchanan 78b). However, the success of inductive methods still rests with a reasonably good set of training instances and an adequate language.

Acknowledgments

This report describes work done in the Computer Science Department at Stanford University. It was supported in part by the Advanced Research Project Agency (Contract DARPA N00039-83-C-0136), the National Institute Health (Grant NIH RR-00785-11), the National Aeronautics and Space Administration (Grant NAG-S-261), the National Science Foundation (Grant IST 83-12148), and Boeing (Grant W266875). We are grateful for the computer time on the SUMEX-AIM resource. We thank Dr. Gabriel Garcia and Dr. Peter B. Gregory for providing us with liver biopsy cases in Stanford Medical Center, which were used in the validation studies.

I. Degree of Certainty

Approaches to inexact reasoning (reasoning under uncertainty) include probabilistic methods (e.g., Bayesian statistical approach used in [Warner 64]), fuzzy set theory [Zadeh 65], CF model [Buchanan and Shortliffe 84], Dempster-Shafer theory ([Shafer 76] and [Barnett 81]). Most of them have been applied to medical decision making. Another approach introduced here, called "degree of certainty", is a variation of the CF model used in MYCIN [Buchanan and Shortliffe 84].

In this work we represent uncertainty with a real number, ranging from -1 to 1, which is much like MYCINs CF except that probability and importance of evidence are kept separate here. Assignment of a degree of certainty to a rule in the knowledge base is based on an expert's estimate, which is the integration of many factors, including probabilistic knowledge and attitude (bias). The degree of certainty is defined as follows:

degree of certainty (h, e) = P(h/e)S(h, e)

where,
- h: hypothesis
- e: evidence, or pattern of attributes (e,)
- P(h/e): probability of h, given e
- S(h, e): semantic importance (strength) of e for h

S(h, e) is a strength factor and may be viewed as a weighting factor assigned to evidence "e" for a given hypothesis "h". In the absence of other information, S(h, e) is proportional to the sum of predictive values of all attributes e_i in the evidence set, e. Predictive value of a single attribute, ranging from 0 to 6, reflects statistical knowledge and importance of that attribute for a given hypothesis.[19]

For confirming evidence,

Mapping:

Σ Predictive value(h, e_i)	S(h,
<2	.4
[2 3)	.6
[3 4)	.8
[4 5)	.9
[5 6)	.95
$\geq$6	1

See [Fu 85] for further details.

Degree of certainty of different pieces of evidence can be combined according to CF combining function in MYCIN [Buchanan and Shortliffe 84] or Dempster-Shafer theory as in [Gordon 84]. Moreover, degree of certainty and CF are related in the following ways. For a confirming evidence "e", if $P(h) \simeq 0$ and we assign S(h, e)=1, then:

$$P(h/e) \simeq CF(h, e) \simeq \text{degree of certainty}(h, e)$$

If P(h) $\not\simeq$ 0, by properly choosing S(h, e), degree of certainty can still approximate CF. The

[19] The assignment of a predictive value for a pattern, based on the predictive values of individual features. is similar to decision making in systems built in EXPERT [Kulikowski and Weiss 82], where combinations of different numbers of major and minor criteria may yield different conclusions. Each criterion is a symptom or a sign or a laboratory test For instance, rheumatic fever can be diagnosed with a pattern of two major criteria or a pattern of one major plus two minor criteria.

main feature of “degree of certainty** is the incorporation of an explicit strength factor which can reflect an expert’s attitude toward evidence (e.g., conservative or aggressive) in a Specific domain.

References

[Barnett 81] Barnett, J. A.
Computational methods for a mathematical theory of evidence.
In ***Proceedings of 7th International Joint Conference on Artificial Intelligence.*** Vancouver, 1981.

[Buchanan 78a] Buchanan, B. G. and Feigenbaum, E. A.
DENDRAL and Meta-DENDRAL: their applications dimension.
Artificial Intelligence 11, 1978.

[Buchanan 78b] Buchanan, B. G. and Mitchell, T.M.
Model-directed learning of production rules.
In Waterman, D. and Hayes-Roth, F. (editor), ***Pattern-Directed Inference systems, .*** Academic Press, New York, 1978.

[Buchanan and Shortliffe 84]
Buchanan, B. G. and Shortliffe, E. H.
Rule-Based Expert Systems.
Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1984.

[Carbonell 83] Carbonell, J. G., Michalski, R. S., Mitchell, T. M.
An Overview of Machine Learning.
In ***Machine Learning,*** chapter 1, . Tioga, Palo Alto, CA, 1983.

[Clancey 83] Clancey, W. J.
The epistemology of a rule-based expert system: A framework for explanation.
Artificial Intelligence 20, 1983.

[Davis and Buchanan 77]
Davis, R., and Buchanan, B. G.
Meta-level knowledge: Overview and applications.
In ***Proceedings of the 5th International Joint Conference on Artificial Intelligence.*** Cambridge, Mass., August, 1977.

[Dietterich 83] Dietterich, T. G., Michalski, R. S.
A Comparative Review of Selected Methods for Learning from Examples.
In ***Machine learning,*** chapter 3, . Tioga Publishing Company, Palo Alto, CA, 1983.

[Fu 85] Fu, Li-Min.
Learning Object-level and Meta-level Knowledge in Expert Systems.
PhD thesis, Stanford University, March, 1985.

[Fu and Buchanan 84]
Fu, Li-Min. and Buchanan, B.G.
Enhancing Performance of Expert Systems by Automated Discovery of Meta-rules.
In ***Proceedings of the 1st Conference on Artificial Intelligence Applications.*** IEEE, Denver, Colorado, December, 1984.

[Gordon 84] Gordon, J. and Shortliffe, E. H.
The Dempster-Shafer Theory of Evidence.
In Buchanan, B. G., Shortliffe, E. H. (editor), ***Rule-Based Expert Systems,*** chapter 13, . Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1984.

[Haggerty 84] Haggerty, J.
Referee and Rulecritic: Two Prototypes for Assessing the Quality of Medical Paper.
Technical Report, Computer Science Department, Stanford University, Master Thesis, 1984.

[Kulikowski and Weiss 82]
Kulikowski, Casimir A., and Weiss, Sholom M.
Representation of expert knowledge for consultation: The CASNET and EXPERT projects.
In Peter Szolovits (editor), ***Artificial Intelligence in Medicine,*** pages 21-55. Westview Press, Boulder, Colo., 1982.

[Langley 83) Langley, P., Bradshaw, G. L., and Simon, H. A.
Rediscovering chemistry with the BACON system.
In ***Machine learning,*** chapter 10, . Tioga Publishing Company, Palo Alto, CA, 1983.

[Lenat 83] Lenat, D. B.
Theory formation by heuristic search. The nature of heuristics II: Background and examples.
Artificial Intelligence 21, 1983.

[Michalski 75] Michalski, R. S.
Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning.
In Rine, D. C. (editor), ***Computer Science and Multiple-Valued Logic Theory and Applications, .*** North-Holland, 1975.

[Michalski 78] Michalski, R. S. and Larson, J. B.
Selection of most representative training examples and incremental generation of VL1 hypotheses: The underlying methodology and description of programs ESEL and AQ11.
Technical Report 867, University of Illinois, 1978.

[Michalski 83a] Michalski, R. S. and Stepp, R. E.
Learning from observations: Conceptual clustering.
In ***Machine learning,*** chapter 11, . Tioga Publishing Company, Palo Alto, CA, 1983.

[Michalski 83b] Michalski, R. S.
Theory and Methodology of Inductive Learning.
In ***Machine Learning,*** chapter ***4,*** . Tioga, Palo Alto, CA, 1983.

[Miller, Pople, and Meyers 82]
Miller, R A., Pople, H.E., Meyers, J D.
INTERNIST-l, an experimental computer-based diagnostic consultant for general internal medicine.
The New England Journal of Medicine 307, 1982.

[Mitchell 78] Mitchell, T. M.
Version Spaces: An approach to concept learning.
PhD thesis, Stanford University, December, 1978.

[Quinlan 83] Quinlan, J. R.
Learning efficient classification procedures and their applications to chess end-games.
In Michalski, R. S. et al (editor), ***Machine Learning,*** chapter 15, . Tioga, Palo Alto, CA, 1983.

[Shafer 76] Shafer, G.
A Mathematical Theory of Evidence.
Princeton University Press, Princeton, NJ, 1976.

[Utgoff 82] Utgoff, P. E.
Acquisition of appropriate bias for inductive concept learning.
Technical Report, Thesis proposal, Department of Computer Science, Rutgers University, 1982.

[Warner 64] Warner, H. R. et al.
Experience with Bayes' Theorem for computer diagnosis of congenital heart disease.
Ann. N. Y. Acad. Sci. 115, 1964.

[Winston 70] Winston, P. H.
Learning structural descriptions from examples.
Technical Report TR-76, Project MAC, MIT, 1970.

[Zadeh 65) Zadeh, L. A.
Fuzzy sets.
Information and Control 8, 1965.