MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo 548 September, 1979

LEARNING DISJUNCTIVE CONCEPTS FROM EXAMPLES

by

Glenn A. Iba

Abstract

This work proposes a theory for machine learning of disjunctive concepts. The paradigm followed is one of teaching and testing, where the teaching is accomplished by presenting a sequence of positive and negative examples of the target concept. The core of the theory has been implemented and tested as computer programs. The theory addresses the problem of deciding when it is appropriate to merge descriptions and when it is appropriate to form a disjunctive split. The approach outlined has the advantage that it allows recovery from overgeneralizations. It is observed that negative examples play an important role in the decision making process, as well as in detecting overgeneralizations and instigating recovery. Because of the ability to recover from overgeneralizations when they occur, the system is less sensitive to the ordering of the training sequence than other systems. The theory is presented in a domain and representation independent format. A few conditions are presented, which abstract the assumptions made about any representation scheme that is to be employed within the theory. The work is illustrated in several different domains, illustrating the generality and flexibility of the theory.

This report describes research done at the Arificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643.

LEARNING DISJUNCTIVE CONCEPTS FROM EXAMPLES

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 Overview

This paper extends the concept learning work of Winston [18] so as to treat a broader range of concepts. The class of concepts dealt with here is that obtained by forming disjunctions of basic conjunctive descriptions. A theory is presented, and implemented as a computer program, that enables the learning of such disjunctive concepts from a presented series of positive and negative examples.

Two central problems are addressed by the theory. The first involves deciding when to merge descriptions and when to split them off to form additional disjunctive terms. The second is the problem of recovering from overgeneralizations. The theory presents a solution to each of these issues. It is found that negative examples play a key role in the solutions presented for each of these problems. The theory has the additional advantage that it is quite robust with respect to the ordering of the teaching sequence. Given sufficient additional examples, the system will converge to a single concept model, independently of the ordering of the initial teaching sequence.

The theory and the implementation are independent of the representation scheme used to describe concepts. A small set of criteria is presented in section 3.1, and any representation scheme satisfying these criteria are suitable for use within the theory (and the implemented system).

Chapter 2 introduces the framework of a general learning system. This abstract system assumes the Winston paradigm of learning from a presented sequence of positive and negative examples. It creates a model for every target concept, and updates the model on the basis of each additional example. This component of the theory is independent of domain, representation, and specific theory of maintaining and updating the concept model. In particular, it is independent of the specific theory for handling disjunctive concepts.

Chapter 3 presents the basic theory for handling disjunctive concepts. This theory is presented and implemented within the general learning system framework presented in Chapter 2. This theory is still independent of domain and representation system. Thus the system should enjoy a potential for wide applicability.

Chapter 4 illustrates the theory in several different domains using various representation schemes. Scenarios II and III are elaborated so as to explicate the functioning of the system. The effects of varying the ordering of the teaching sequence are discussed.

Chapter 5 extends the basic theory to allow symmetric treatment of positive and negative examples. Under this scheme, the system is simultaneously building a model of the target concept

and a model of its negation. The interactions of the two models are explored.

Chapter 6 presents a version of the theory which relaxes some of the memory requirements assumed in the basic and symmetric versions of the theory. A typical time-space tradeoff is encountered. The system can get by with smaller memory cost, but only at the risk of taking longer to attain the desired model of the target concept.

Chapter 7 reviews the current state of the implementation of the system described in the preceding chapters.

Chapter 8 concludes with a criticism of the theory. Comparison is made with other existing work. Some problems for additional work are proposed.

1.2 The Phenomenon of Disjunctive Concepts

There are certain concepts which cannot be described or defined in terms of a simple conjunction of features. At some level they seem to require the use of an "or" or disjunction, hence I label them "disjunctive concepts". Their form is more like: <conjunctive-description-1> OR <conjunctive-description-2>. More generally there can be any finite number of conjunctive terms in the disjunctive combination. This form is analogous to the "sum of products" form for boolean functions.

Some examples are in order. In the blocks world the concept of PRISM, defined as a BRICK or a WEDGE (but not a PYRAMID for example), is a disjunctive concept. In the kinship relations domain the relation UNCLE-OF is disjunctive, either (BROTHER-OF a PARENT-OF) OR (HUSBAND-OF a SISTER-OF a PARENT-OF). There is no way to merge the two conjunctive descriptions into a single conjunctive description without the aid of an auxiliary concept to hide the disjunctiveness within it (eg, defining EXTENDED-BROTHER as BROTHER OR BROTHER-IN-LAW). In the domain of orthoganal grids the concept of ORTHOGONAL-DIRECTION can be expressed as NORTH or SOUTH or EAST or WEST.

1.3 The Problem of Learning Disjunctive Concepts

The question arises as to how a concept learning system might deal with disjunctive concepts. Winston's system could not learn such concepts because it represented each concept as a single conjunctive description. Every new example could lead to changes or updates of this model, but none could ever lead to a disjunctive splitting into two or more conjunctive descriptions.

Winston's system did represent certain disjunctive concepts such as PRISM, but these were implicitly described in terms of AKO links (eg BRICK and WEDGE were both AKO

PRISM). Such concepts were built into the initial memory structure. There was no facility to enable these concepts to be learned in terms of already available concepts.

Where the possibility exists of encountering disjunctive concepts, there are two possible ways of handling a new positive example. Either the new example can be merged into the previous model, or it can be split to form a new (additional) branch of a disjunction. The problem of handling disjunctive concepts can be reformulated as the problem of deciding whether merging or splitting is appropriate.

Another problem that can arise is that of overgeneralization. Winston's learning system [18] was a conservative generalizer. It moved cautiously only in the direction of greater generality. Its basic generalizing mechanism was a merging process which combined networks. If presented by two positive examples taken from separate branches of a disjunctive concept, this system would apply a merge, resulting in an over-general model. Winston's program lacked the ability to use negative examples to recover from such over-generalizations. If a system for handling disjunctive concepts is to be flexible in terms of permissible sequencing of teaching examples, then it must be capable of recovering from overgeneralizations.

1.4 Illustrative Scenarios

Three example scenarios will serve to illustrate the behavior of the program which implements the theory presented in this paper. The first demonstrates the basic ability to recover from overgeneralization and to form disjunctions. The second gives a more complex example, still using feature sets as the representation. The third shows the system using a network representation scheme to learn a relational concept.

1.4.1 Scenario I

The first scenario is for the target concept of PRISM defined as having either the feature BRICK or the feature WEDGE. Here the example (PYRAMID) will serve as a negative example. Note that for this illustration examples are presented as sets of features enclosed in parentheses. Positive examples are indicated by POSITIVE and negative examples by NEGATIVE.

Example I. PRISM: (BRICK) POSITIVE

This causes the system to create the model (BRICK). If the system were given some test questions it would classify (RED BRICK), (BRICK), and (SMALL THIN GREEN BRICK) all as examples

of PRISM. It would judge (PYRAMID), (GREEN WEDGE), (WEDGE), and (LARGE BLUE WEDGE) as not being examples of PRISM.

Example 2. PRISM: (WEDGE) POSITIVE

This example leads to the overgeneralization ( ), the empty feature set, which means that everything is considered to be a PRISM. Any test questions at all at this point would result in acceptance as examples of PRISM.

Example 3. PRISM: (PYRAMID) NEGATIVE

On the basis of this negative example, the system recovers from its overgeneralization and arrives at the desired model: (OR (BRICK) (WEDGE)). Testing at this point, yields the desired correct behavior: (RED BRICK), (THIN SMALL WEDGE), (BRICK), (LARGE BLUE THICK WEDGE) are all classified as PRISM's; and (PYRAMID), (LARGE GREEN PYRAMID), (DODECAHEDRON) would be rejected as non-PRISM's.

1.4.2 Scenario II

The target concept here is RED-OR-TRIANGLE, that is, anything that either has the feature RED or the feature TRIANGLE. Examples are again presented as sets of features.

Example 1. RED-OR-TRIANGLE: (RED SQUARE THIN LARGE) POSITIVE
Resulting model: (RED SQUARE THIN LARGE)

Example 2. RED-OR-TRIANGLE: (BLUE TRIANGLE THIN SMALL) POSITIVE
Resulting model: (THIN)

Example 3. RED-OR-TRIANGLE: (BLUE CIRCLE THIN SMALL) NEGATIVE
Resulting model: (OR (BLUE TRIANGLE THIN SMALL)(RED SQUARE THIN LARGE))

Example 4. RED-OR-TRIANGLE: (RED SQUARE THICK SMALL) POSITIVE
Resulting model: (OR (BLUE TRIANGLE THIN SMALL)(RED SQUARE))

Example 5. RED-OR-TRIANGLE: (GREEN THICK LARGE TRIANGLE) POSITIVE
Resulting model: (OR (TRIANGLE)(RED SQUARE))

Example 6. RED-OR-TRIANGLE: (RED CIRCLE THIN LARGE) POSITIVE
Resulting model: (OR (TRIANGLE)(RED))

This scenario will be explained in much greater detail in section 4.2.

1.4.3 Scenario III

In this scenario, the target concept is UNCLE-NESS, which can be represented disjunctively as either a brother of a parent, or a husband of a sister of a parent. The three examples used in this scenario are represented as networks of nodes and relations:

```
<UNCLE-1> ::=
        ((NODES (A B C))
         (LINKS (AKO A PERSON)
                (AKO B PERSON)
                (AKO C PERSON)
                (MALE C)
                (B PARENT-OF A)
                (C BROTHER-OF B)))

<UNCLE-2> ::=
        ((NODES (A B C D))
         (LINKS (AKO A PERSON)
                (AKO B PERSON)
                (AKO C PERSON)
                (AKO D PERSON)
                (MALE D)
                (B PARENT-OF A)
                (C SISTER-OF B)
                (D HUSBAND-OF C)))
```

<NON-UNCLE> ::=

((NODES (A B C))
(LINKS (AKO A PERSON)
(AKO B PERSON)
(AKO C PERSON)
(MALE C)
(B PARENT-OF A)
(C PARENT-OF B)))

Example 1. UNCLE-NESS: <UNCLE-1> POSITIVE
Resulting model: <UNCLE-1>

Example 2. UNCLE-NESS: <UNCLE-2> POSITIVE
Resulting model: <DESCR-1>
where <DESCR-1> ::=

((NODES (A B C))
(LINKS (AKO A PERSON)
(AKO B PERSON)
(AKO C PERSON)
(B PARENT-OF A)
(MALE C))).

Note that this is an incorrect overgeneralization.

Example 3. UNCLE-NESS: <NON-UNCLE> NEGATIVE
Resulting model: (OR <UNCLE-1> <UNCLE-2>)

Again the presentation of a negative example leads to recovery from an overgeneralization, and results in a disjunction. This scenario will be discussed in greater detail in section 4.4.6.

2.0 A FRAMEWORK FOR A GENERAL CONCEPT LEARNING SYSTEM

2.1 Overview and description of performance

In this section a presentation will be given of a concept learning system in an abstracted form. In the subsequent section certain parameters of this system will be specified to provide a theory for learning disjunctive concepts.

The present framework for a learning system follows the traditional teaching/testing paradigm of Winston [18]. Teaching takes place by the presentation of a sequence of positive and negative examples of the concept to be learned. Learning is demonstrated by testing the system with new (or old) examples of the concepts it is expected to have learned.

More specifically the system performs two operations: LEARN and TEST. The LEARN operator takes three arguments: name example value. The name argument can be any string and serves only as an identifier for the concept. The example argument is a description of an object or situation. It must be described in some description language. This description language is one parameter of the general learning system. The value is either POSITIVE or NEGATIVE, indicating whether the example represents a positive or negative instance of the concept. The LEARN operator causes the system to update the model based on the newly presented example, or to create a new model if no model is currently known for the named concept. The functional definition of LEARN is left open, thus this procedure constitutes one of the parameters of the learning-system.

The TEST operator takes two arguments: name example. These are the concept name and a description, respectively, as in the case of LEARN. The TEST operator returns as its value either YES or NO, according to whether or not the system expects the given description is or is not an example, based on its current model of the concept. It makes this prediction by calling a function SATISFIES-MODEL? which takes two arguments: description model. This predicate returns TRUE if the description argument satisfies the concept model argument, otherwise FALSE. The SATISFIES-MODEL? predicate is another parameter of the learning system. Its definition must be provided in order to use the system. The definition of the TEST operator is fixed, however, and therefore is not a parameter of the learning system.

The system has a capability to learn as a result of its testing activities. After responding to a TEST command, the system asks the teacher for feedback. The teacher can then tell the system whether or not the prediction was correct. The system computes the true value of the

current example (simply the predicted value if the feedback was positive, or the negation of the predicted value in case of negative feedback). Based on this correct value, the system updates its model of the concept just as if the command had been

(LEARN CONCEPT-NAME CURRENT-EXAMPLE CORRECTED-VALUE). By this simple means the system can learn from its mistakes.

2.2 Parameters of the system

The following are parameters of the general learning system, that is, they are left open in designing the general system but must be specified or defined in order to completely specify a particular learning system.

2.2.1 Descriptions and concept models

It is necessary for the system to have some way of describing examples. Therefore a representation scheme must be provided. Associated with the description scheme must be a predicate operator SATISFIES-DESCRIPTION? which can determine of a given example whether it satisfies a given description.

The system must also have a convention for the structure of models of concepts. This structure may be different than the convention for representing descriptions, for example, it could be a set or list of descriptions. Thus a second predicate, SATISFIES-MODEL?, must be provided to determine whether an example satisfies a particular model. This predicate may make use of SATISFIES-DESCRIPTION? in making its determination.

2.2.2 Memory

The system must have a memory facility, herein called the CONCEPT-MEMORY, for storing concept models for later retrieval. The operators associated with this are STORE-CONCEPT and RETRIEVE-CONCEPT. STORE-CONCEPT takes two arguments: a concept name and a concept model. It creates an association of the given name and the given model in CONCEPT-MEMORY, so that subsequent calls to RETRIEVE-CONCEPT with the same concept name will return the given concept model. RETRIEVE-CONCEPT takes a single argument: the name of the concept to be retrieved. It returns the current concept model corresponding to that concept name if it exists, otherwise it returns CONCEPT-UNDEFINED.

2.2.3 The LEARN operator

A definition must be provided for the operator LEARN mentioned above in section 2.1. Its three arguments are: name example value. This operator is responsible for retrieving the old concept model (if it exists), creating a new model based on the old model and the new example, and storing the new model for the concept in CONCEPT-MEMORY.

SUMMARY OF SYSTEM PARAMETERS:

Data Structures:

CONCEPT-MEMORY

Data Types:

concept model

description

Predicates:

SATISFIES-MODEL?

SATISFIES-DESCRIPTION?

Operators:

STORE-CONCEPT

RETRIEVE-CONCEPT

LEARN

3.0 A BASIC THEORY FOR HANDLING DISJUNCTIVE CONCEPTS

3.1 Domains to which the theory applies

The present theory of learning disjunctive concepts is applicable to a broad range of domains. There are just a few assumptions and constraints that must be satisfied. Any domain with a concept representation scheme that satisfies these constraints is suitable for use in this theory.

The basic constraints that must be satisfied by the representation scheme are:

1. The set of possible descriptions is partially-ordered according to increasing generality by a binary predicate called GREATER-GENERALITY?. (GREATER-GENERALITY? A B) is true if and only if the description A is at least as general as the description B. [Intuitively, anything described by B could also be described by A]. Note that (GREATER-GENERALITY? A A) is assumed to be true since a concept is at least as general as itself.
2. There is an operation MERGE that takes two descriptions as arguments and returns a description which is at least as general as each argument in turn. Thus (GREATER-GENERALITY? (MERGE A B) A) and (GREATER-GENERALITY? (MERGE A B) B) are always true.
3. An additional assumption is made about the structure of the concept space: that it is MONOTONIC with respect to positive examples. Under this assumption, if a given description is a positive example of a concept, then so also is every description that is more specific (less general) than the given description.

As an illustration of these ideas consider the very simple ATTRIBUTES-WORLD in which descriptions are merely finite sets of features or attributes. Some representative feature sets are (RED SQUARE), (RED), (LARGE RED), (SQUARE), and (SMALL BLUE CIRCLE). A description set is satisfied by those objects which possess all the features of the set.

The GREATER-GENERALITY? predicate corresponds to the "subset" relation applied to sets. For example (GREATER-GENERALITY? (RED) (RED SQUARE)) is true since (RED) is a subset of (RED SQUARE).

The MERGE operation is performed by intersecting the two arguments. Since the intersection of two sets is a subset of each of them, condition 2 above is satisfied. The result of (MERGE (RED SQUARE) (LARGE RED)) is the description set (RED), which has greater

generality than both (RED SQUARE) and (LARGE RED).

It must be emphasized that this simple domain is merely used for illustration because of its clarity and simplicity, and that much richer representations such as frames or networks can be used by defining suitable GREATER-GENERALITY? and MERGE functions. Any representation scheme that can be made to satisfy the specified conditions is valid for the purposes of the theory.

3.2 Presentation of the theory

3.2.1 Overview

This theory solves the problem (of deciding whether to merge or split) by assuming that merging is the appropriate choice unless evidence explicitly indicates otherwise. This may lead to incorrect generalizations in some teaching sequences, but the system has the ability to recover from these when later evidence points to a contradiction. Such over-generalization is detected when a negative example satisfies a description resulting from a merge. The response in such a case is to split up the merged description into a disjunction of descriptions, each of which is compatible with the negative example. This is possible because the system maintains a record of the positive examples that contributed to the merge. A record is also maintained of the negative examples encountered. These serve to inhibit overgeneralizations that might otherwise result from subsequent positive examples.

3.2.2 Data Structures and Types

3.2.2.1 The concept model

The concept model will be structured into two parts - a POSITIVE-PART and a NEGATIVE-PART. The POSITIVE-PART will be a set of CLUSTERS. Clusters are structured objects with two parts - DESCRIPTION and EXAMPLES. The EXAMPLES part is a list of one or more descriptions, and the DESCRIPTION part is a description-set obtained by MERGEing all of the descriptions in the EXAMPLES part to obtain a description that is more general than each of them. Since MERGE may fail to be associative, assume a left to right order of application. In case of a single example description the result is just that description, no MERGEs being necessary. The NEGATIVE-PART of the concept model will merely be a set of descriptions.

In summary:

CONCEPT-MODEL ::= POSITIVE-PART NEGATIVE-PART
POSITIVE-PART ::= CLUSTER-SET
CLUSTER-SET ::= set with elements of type CLUSTER
CLUSTER ::= DESCRIPTION-PART EXAMPLES-PART
DESCRIPTION-PART ::= DESCRIPTION
EXAMPLES-PART ::= set with elements of type DESCRIPTION
NEGATIVE-PART ::= set with elements of type DESCRIPTION

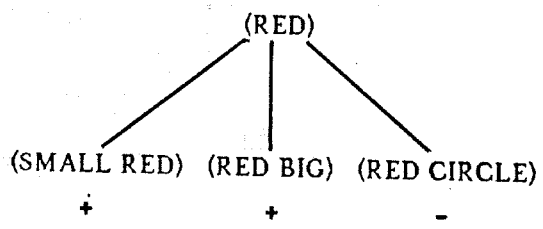
The positive-part of the model corresponds to a disjunctive form in which the clusters correspond to terms. The concept can be considered to be (OR CLUST-DESCR-1 CLUST-DESCR-2 ... CLUST-DESCR-n) where CLUST-DESCR-i is just the description part of the i-th cluster. The examples that contribute to a cluster are kept around since they may be needed at a later time. Specifically, it may turn out that a cluster is overly general. In such a case, it is necessary to split the cluster up, and the constituent examples are quite helpful in this recovery process.

It is the intention that every known positive example of the concept will be represented in (at least) one cluster of the cluster-set. The clusters serve to group the positive examples into MAXIMAL, CONSISTENT groups. In testing a new description to see if it satisfies the model, an affirmative result is obtained if there is any cluster whose description-part has greater-generality than the description being tested. This is where the disjunctive nature of the cluster set is evidenced.

The clusters must be CONSISTENT with the known negative examples of the concept. If the description-part of a cluster has GREATER-GENERALITY than any one of the known negative examples, then there is a contradiction. The description from the cluster would be judged a positive example by the SATISFIES-MODEL test. Then by the MONOTONICITY condition (assumption 3 of section 3.1) the negative example, being less general than the given description, should also be positive. Such a contradiction will be termed an INCONSISTENCY. Inconsistency results from over-generalization. By maintaining the consistency of the clusters, this theory uses negative examples to inhibit overgeneralization (and to recover from it when it happens due to ordering of the teaching sequence).

The clusters will also be MAXIMAL with respect to the negative examples. A cluster is said to be MAXIMAL when no other cluster description can be MERGEd into it without violating the CONSISTENCY condition. This condition insures that it is really necessary to have splits into

as many clusters as are present. If two clusters can be combined (by intersecting their descriptions) without violating the consistency condition, then they are not maximal.



Consider the above diagram as an example. If the positive examples (SMALL RED) and (RED BIG) were merged to form a single cluster with description (RED), there would be an inconsistency (since (RED CIRCLE) contradicts the theory that anything red is an instance or the concept). The negative example forces a disjunctive split of the two positive examples into separate clusters. These clusters, consisting of just one example each, are also maximal, since their combination would lead to the inconsistency just cited.

The negative-part of the model serves simply to record the negative examples of the concept that have been presented thus far. They are needed in order to prevent future over-generalization as a result of positive examples. This should all be clearer when the treatment of examples by the system is dealt with.

Associated with concept models is the SATISFIES-MODEL? predicate. For the structured concept model described above, the SATISFIES-MODEL? predicate is defined by the following procedure. First the set of descriptions is collected from the clusters forming the positive part of the model. Then for each description (SATISFIES-DESCRIPTION? example description) is computed. The value returned is just the OR of these computed values. So an example satisfies such a disjunctive model if and only if it satisfies at least one of the cluster descriptions in the positive part of the model.

3.2.2.2 Concept Memory

CONCEPT-MEMORY is implemented trivially as a list of pairs, each pair consisting of a concept name and a concept model. There should only be one pair containing any given name. The RETRIEVE-CONCEPT function simply searches for a pair whose name matches the argument supplied. It returns as its value the concept model from this pair. If no pair is found then the value CONCEPT-UNDEFINED is returned. The STORE-CONCEPT function finds

the pair corresponding to its name argument and substitutes the new concept model for the old model. If no pair is found with the given name then a new pair is formed from the name and model supplied, and this pair is added to the list.

3.2.3 The LEARN operator

3.2.3.1 Summary

The Learn operator takes three arguments: name example value. These were described in section 2.1. This operator retrieves the old concept model (if one exists) corresponding to the name supplied. It then updates this model based on the information of the new example. If no previous model exists, then a new model is constructed based on the new example. Finally, the updated model is stored in memory as the current model of the named concept. The core of this operator is the procedure for updating the concept model. This will be described in the next section.

3.2.3.2 Updating the model

3.2.3.2.1 Treatment of positive examples

When the value argument of LEARN indicates a positive example of a concept, the LEARN procedure tries to add the example into each cluster of the positive part of the concept model. It does this by merging the description of the example with the description from the cluster. If this merge is satisfied by any of the known negative examples, then the example is incompatible with the cluster, that is, an inconsistency would result. In such a case the cluster is left unchanged (retaining the description it had prior to the consideration of the merge). If, on the other hand, the resultant description is not satisfied by any known negative example, then the example is consistent with the cluster and will be added to it. This addition consists of

(1) replacing the description part of the cluster by the new (merged) description, and

(2) adding the new example to the set of examples in the examples part of the cluster.

If the positive example is compatible with more than one cluster, it will be added to each of them. If there is no cluster with which the example is compatible, then the example will be "split" off to form a cluster of its own. This new cluster will take its description part from the example's description, and its example part will contain just one element - the example itself.

It can be readily observed that these procedures maintain the MAXIMALITY and CONSISTENCY conditions on the clusters.

3.2.3.2.2 Treatment of negative examples

When a negative example is encountered, the system undertakes two actions. (1) It adds the example to the negative part of the model. (2) It checks each of the clusters in the positive part of the model to see if any are inconsistent with the new negative example. Any clusters that are inconsistent must be split up into a set of clusters which are maximal, consistent, and which include among them all of the examples of the original cluster.

To do this, a new cluster set is formed by adding the positive examples of the old cluster one at a time to a null cluster set, subject to the constraint of the negative example in question. The addition of the examples to the cluster set is done following the procedure outlined above for handling positive examples. The clusters in this newly formed set replace the old inconsistent cluster in the positive part of the model.

This process of breaking up clusters may result in clusters that are no longer maximal with respect to the set of negative examples, (though they will all be consistent). This is because certain of the clusters arising from the splitting process may be MERGE-able with other clusters in the cluster set. Thus as a final step the entire cluster set is examined for possible simplifications through the combining of clusters. The simplified cluster set replaces the old cluster set in the positive part of the concept model.

3.2.4 Remaining parameters of the theory

By specifying some of the parameters of the general learning framework I have specialized it to a theory for learning disjunctive concepts. In doing so, I have not explicitly specified the particular representation scheme that is to be used. Any representation scheme satisfying the criteria of section 3.1 can be used. Thus the final parameters are the format for description and representation, the SATISFIES-DESCRIPTION? predicate, the GREATER-GENERALITY? predicate, and the MERGE operator. These must be supplied for a particular domain in order to apply the theory. Some illustrations will be given in the next chapter.

SUMMARY OF DISJUNCTIVE THEORY PARAMETERS:

DESCRIPTION SCHEME
SATISFIES-DESCRIPTION?
GREATER-GENERALITY?
MERGE

4.0 APPLICATION AND ILLUSTRATION OF THE BASIC THEORY

4.1 Feature Sets and the Attributes World

In order to illustrate the basic functioning and capabilities of the theory, I introduce a particularly simple domain called ATTRIBUTES WORLD. Examples and descriptions in this world consist merely of sets of features or attributes. The name is motivated by the set of learning toys called "attribute blocks", which are a set of plastic blocks with varying shape, color, size, and thickness. In schools these blocks are frequently used to teach concepts of set theory. A block might be described as (LARGE RED THIN TRIANGLE). The features are treated as a set, so the order of features in the list representation is not significant. In its more general form, an example in ATTRIBUTES WORLD can be described as an arbitrary set of independent features.

The SATISFIES-DESCRIPTION? and the GREATER-GENERALITY? predicates will both be implemented as the boolean predicate testing for set containment. The forms (SATISFIES-DESCRIPTION? A B) and (GREATER-GENERALITY? B A) are specified to be true if A contains B (that is, B is a subset of A) where A and B are feature sets. Note the reversed order of the arguments - B is more general than A implies that description A satisfies description B, that is, anything satisfying the first description will also satisfy the second.

The MERGE operator will be implemented as set intersection. Thus (MERGE A B) will return those features shared by A and B. The intersection is contained in each set individually, thus satisfying the condition that the merged description be more general than each of the sets being merged. Refer to section 3.1.

4.2 A sample teaching sequence

Consider the following illustrative teaching sequence for the concept "RED-OR-TRIANGLE":

1. (LEARN RED-OR-TRIANGLE (RED SQUARE THIN LARGE) POSITIVE)
2. (LEARN RED-OR-TRIANGLE (BLUE TRIANGLE THIN SMALL) POSITIVE)
3. (LEARN RED-OR-TRIANGLE (BLUE CIRCLE THIN SMALL) NEGATIVE)
4. (LEARN RED-OR-TRIANGLE (RED SQUARE THICK SMALL) POSITIVE)
5. (LEARN RED-OR-TRIANGLE (GREEN THICK LARGE TRIANGLE) POSITIVE)
6. (LEARN RED-OR-TRIANGLE (RED CIRCLE THIN LARGE) POSITIVE)

On being presented with the first example the system tries to look up the concept

RED-OR-TRIANGLE. No model is currently known, so a new one is created. The positive part will consist of one cluster whose description and sole example will be (RED SQUARE THIN LARGE). The negative part will be empty. Thus after step 1 the concept model structure appears as follows:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(RED SQUARE THIN LARGE)
EXAMPLES
(RED SQUARE THIN LARGE)
NEGATIVE-PART

The second example is also positive so the system tries to add it to each cluster in the positive part of the model. There is only one cluster at this time. The example is merged (intersected) with the description of the cluster. (MERGE (BLUE TRIANGLE THIN SMALL) (RED SQUARE THIN LARGE)) is evaluated yielding the description (THIN). Since there are no negative examples (the negative part of previous model is empty) to contradict this description (consistency condition), the example may be added to the cluster. Thus the new description is (THIN) and both examples are listed. The resulting concept structure is:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(THIN)
EXAMPLES
(BLUE TRIANGLE THIN SMALL)
(RED SQUARE THIN LARGE)
NEGATIVE-PART

At this stage, "thin" is the essential model held by the system. Anything having the feature THIN

will be viewed by the system as satisfying this model. This illustrates how the system performs as a simple conjunctive generalizer in the absence of evidence "forcing" a disjunction.

The third example is a negative example. It indicates to the system that overgeneralization has occurred and that a disjunctive split is necessary. The system compares the description of the negative example to the description of each cluster. It notes that the cluster description (THIN) is satisfied by the negative example (BLUE CIRCLE THIN SMALL). Thus it is necessary to split up the cluster (as it violates the consistency condition). Two separate clusters are formed from the examples of the old cluster. These clusters, in turn, replace the old cluster. Finally the negative example is added to the negative-part of the model. After the third example:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(BLUE TRIANGLE THIN SMALL)
EXAMPLES
(BLUE TRIANGLE THIN SMALL)
CLUSTER
DESCRIPTION
(RED SQUARE THIN LARGE)
EXAMPLES
(RED SQUARE THIN LARGE)
NEGATIVE-PART
(BLUE CIRCLE THIN SMALL)

At this point the model may interpreted as "either a (thin small blue triangle) or a (large thin red square)".

The fourth example illustrates the treatment of a positive example in the case of a non-trivial disjunction of clusters. It also demonstrates how knowledge of negative examples can inhibit over-generalization. The system tries to add the example to each cluster. With the first cluster,

(MERGE (RED SQUARE THICK SMALL) (BLUE TRIANGLE THIN SMALL)) yields the description (SMALL). This description is satisfied by the negative example (BLUE CIRCLE THIN SMALL), so the example cannot be added to the first cluster. With the second cluster, (MERGE (RED SQUARE THICK SMALL) (RED SQUARE THIN LARGE)) yields the description (RED SQUARE). This description is consistent with all the known negative examples (there is only one so far), so the example can be added to the second cluster. The result is:

```
"RED-OR-TRIANGLE"
        POSITIVE-PART
                CLUSTER
                        DESCRIPTION
                                (BLUE TRIANGLE THIN SMALL)
                        EXAMPLES
                                (BLUE TRIANGLE THIN SMALL)
                CLUSTER
                        DESCRIPTION
                                (RED SQUARE)
                        EXAMPLES
                                (RED SQUARE THICK SMALL)
                                (RED SQUARE THIN LARGE)
        NEGATIVE-PART
                (BLUE CIRCLE THIN SMALL)
```

In words "either a small thin blue triangle or a red square".

The fifth example works in a similar way to loosen restrictions on the first cluster, resulting in the model:

```
"RED-OR-TRIANGLE"
        POSITIVE-PART
                CLUSTER
                        DESCRIPTION
```

(TRIANGLE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)
(BLUE TRIANGLE THIN SMALL)
CLUSTER
DESCRIPTION
(RED SQUARE)
EXAMPLES
(RED SQUARE THICK SMALL)
(RED SQUARE THIN LARGE)
NEGATIVE-PART
(BLUE CIRCLE THIN SMALL)

"either a triangle or a red square"

The attempt to add (GREEN THICK LARGE TRIANGLE) to the second cluster failed because the merge (intersection) led to an empty feature set. The empty feature set is satisfied by any other feature set, and in particular by the negative example (BLUE CIRCLE THIN SMALL).

The sixth and final example, (red circle thin large), removes the square constraint from the second cluster. The final model appears:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(TRIANGLE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)
(BLUE TRIANGLE THIN SMALL)
CLUSTER
DESCRIPTION
(RED)

EXAMPLES

(RED CIRCLE THIN LARGE)

(RED SQUARE THICK SMALL)

(RED SQUARE THIN LARGE)

NEGATIVE-PART

(BLUE CIRCLE THIN SMALL)

"either a triangle or something red"

4.3 Variations in order of the teaching sequence

One of the features of this theory of learning disjunctive concepts is that it is less dependent on the order of presentation of examples than was Winston's system. This is due to its ability to recover from overgeneralization. Winston's system did not derive any information from a negative example if it was the first item in the teaching sequence. In addition, it was necessary to carefully order the teaching sequence so as not to mislead the program. The currently proposed system for disjunctive concepts does not suffer these restrictions.

There is some dependency on the specific ordering of the teaching sequence, but it is not irrevocable. Given two teaching sequences, one a permutation of the other, they may result in slightly different models. With the presentation of an appropriate sequence of additional examples, the models will converge.

4.3.1 An alternative ordering of the sample teaching sequence

In this section, an alternative ordering will be considered for the teaching sequence presented in section 4.2. Assume that the negative example from that sequence were not presented until after all of the positive examples. The sequence would then appear as:

RED-OR-TRIANGLE:

1. (LEARN RED-OR-TRIANGLE (RED SQUARE THIN LARGE) POSITIVE)
2. (LEARN RED-OR-TRIANGLE (BLUE TRIANGLE THIN SMALL) POSITIVE)
3. (LEARN RED-OR-TRIANGLE (RED SQUARE THICK SMALL) POSITIVE)
4. (LEARN RED-OR-TRIANGLE (GREEN THICK LARGE TRIANGLE) POSITIVE)
5. (LEARN RED-OR-TRIANGLE (RED CIRCLE THIN LARGE) POSITIVE)
6. (LEARN RED-OR-TRIANGLE (BLUE CIRCLE THIN SMALL) NEGATIVE)

Through step 5 the system would conjunctively accumulate the examples in a single cluster, there not being any negative examples to indicate a need for disjunctive splitting. The model after the first five examples would be:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
()
EXAMPLES
(RED CIRCLE THIN LARGE)
(GREEN THICK LARGE TRIANGLE)
(RED SQUARE THICK SMALL)
(BLUE TRIANGLE THIN SMALL)
(RED SQUARE THIN LARGE)
NEGATIVE-PART
"anything"

At this point the system believes that anything satisfies the concept. This is not unreasonable since it has yet to encounter a negative example. This is a typical case of overgeneralization. Note that the cluster description formed by merging the examples is empty, since there are no features shared by all five positive examples.

When the system is given a negative example in step 6, it successfully recovers from the overgeneralization, arriving at a model similar to that attained through the different teaching sequence discussed in section 4.2. The recovery process is instigated by discovery that the new negative example satisfies the cluster description of the one cluster in the positive part of the old concept model. This causes the cluster to be broken up to re-establish consistency of the clusters with the negative example.

The procedure for splitting up is called REBUILD-CLUSTER. This procedure takes two arguments: cluster and negative-example. The task of REBUILD-CLUSTER is to restructure

the examples from the cluster into a set of clusters which are maximal and consistent with respect to the negative-example. It does this by taking the examples from the example set of the cluster being rebuilt, and inserting them one by one into ANSWER, which is initially an empty cluster-list. This is done by the INSERT-IN-CLUSTER-LIST procedure which adds an example to a set of clusters, subject to the constraint of a single negative example. In this case the negative example is the same one that initiated the call to REBUILD-CLUSTER (and was passed to it as the second argument). INSERT-IN-CLUSTER-LIST insures that the cluster-list stays maximal and consistent (with respect to the negative example) with the insertion of each additional example.

REBUILD-CLUSTER returns the value of ANSWER, which is a cluster list of maximal, consistent clusters, which include among them all the examples of the input cluster. The clusters of the cluster list which is returned are substituted for the single overgeneral cluster.

Taking the concrete example at hand, there is one cluster:

CLUSTER
DESCRIPTION
()
EXAMPLES
(RED CIRCLE THIN LARGE)
(GREEN THICK LARGE TRIANGLE)
(RED SQUARE THICK SMALL)
(BLUE TRIANGLE THIN SMALL)
(RED SQUARE THIN LARGE)

and the negative example:

(BLUE CIRCLE THIN SMALL). Because the negative example satisfies the (empty) description of the cluster, it is necessary to rebuild the cluster. ANSWER is initialized to the empty cluster list. Then the first example, (RED CIRCLE THIN LARGE), is added subject to the constraint of the negative example. This first example forms a single cluster, resulting in:

ANSWER
CLUSTER
DESCRIPTION
(RED CIRCLE THIN LARGE)

EXAMPLES

(RED CIRCLE THIN LARGE)

The second example, (GREEN THICK LARGE TRIANGLE), is merged with the description of the cluster in ANSWER to yield (LARGE). This is compatible with the negative example, that is, (BLUE CIRCLE THIN SMALL) does not satisfy (LARGE), so the second example is added to the cluster, yielding:

ANSWER

CLUSTER

DESCRIPTION

(LARGE)

EXAMPLES

(GREEN THICK LARGE TRIANGLE)

(RED CIRCLE THIN LARGE)

The third example, (RED SQUARE THICK SMALL), when merged with (LARGE), yields the empty description (). This is inconsistent since the negative example satisfies it, thus this third example cannot be added to the cluster. Instead it goes to start a new cluster, resulting in:

ANSWER

CLUSTER

DESCRIPTION

(LARGE)

EXAMPLES

(GREEN THICK LARGE TRIANGLE)

(RED CIRCLE THIN LARGE)

CLUSTER

DESCRIPTION

(RED SQUARE THICK SMALL)

EXAMPLES

(RED SQUARE THICK SMALL)

The fourth example, (BLUE TRIANGLE THIN SMALL), fails to be added to the first cluster since the merged description would be (), which is inconsistent. It fails to be added to the second cluster, since the merged description, (SMALL), is inconsistent with the negative example (BLUE CIRCLE THIN SMALL). Since the fourth example cannot be added to either of the two existing clusters, it will be added to the cluster set as a new cluster, resulting in:

ANSWER
CLUSTER
DESCRIPTION
(LARGE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)
(RED CIRCLE THIN LARGE)
CLUSTER
DESCRIPTION
(RED SQUARE THICK SMALL)
EXAMPLES
(RED SQUARE THICK SMALL)
CLUSTER
DESCRIPTION
(BLUE TRIANGLE THIN SMALL)
EXAMPLES
(BLUE TRIANGLE THIN SMALL)

Finally, the fifth example, (RED SQUARE THIN LARGE), is added. It can be added to the first cluster since it merges with (LARGE) to yield (LARGE). It can also be added to the second cluster, since the merge is (RED SQUARE). It fails on the third cluster since the merge would be (BLUE THIN), which is contradicted by the negative example (BLUE CIRCLE THIN SMALL). Thus the new cluster set is:

ANSWER
CLUSTER
DESCRIPTION

(LARGE)
EXAMPLES
(RED SQUARE THIN LARGE)
(GREEN THICK LARGE TRIANGLE)
(RED CIRCLE THIN LARGE)
CLUSTER
DESCRIPTION
(RED SQUARE)
EXAMPLES
(RED SQUARE THIN LARGE)
(RED SQUARE THICK SMALL)
CLUSTER
DESCRIPTION
(BLUE TRIANGLE THIN SMALL)
EXAMPLES
(BLUE TRIANGLE THIN SMALL)

This is the final cluster set. Each of the examples is now represented in at least one of the clusters. This new cluster set restores consistency of the clusters with the negative examples. The model generating procedure now replaces the old inconsistent cluster by this new set of clusters to yield the model:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(LARGE)
EXAMPLES
(RED SQUARE THIN LARGE)
(GREEN THICK LARGE TRIANGLE)
(RED CIRCLE THIN LARGE)
CLUSTER
DESCRIPTION

(RED SQUARE)
EXAMPLES
(RED SQUARE THIN LARGE)
(RED SQUARE THICK SMALL)
CLUSTER
DESCRIPTION
(BLUE TRIANGLE THIN SMALL)
EXAMPLES
(BLUE TRIANGLE THIN SMALL)
NEGATIVE-PART
(BLUE CIRCLE THIN SMALL)

"either large or a red square or a small thin blue triangle"

The last step of the model generating process is to check the clusters for maximality. The clusters are maximal as they stand, since the merging of any two of the clusters of the positive part would result in the empty description (), which would be contradicted by the known negative example. Since the clusters check out as maximal, the above model is returned unchanged as the final model.

4.3.2 Convergence of models after an additional example

As a result of the variation in the ordering teaching of the teaching sequence, the models are slightly different. In particular, the second model has a spurious cluster corresponding to the term "large" in the disjunction. One additional example, however, suffices to bring the models to convergence.

Suppose the negative example (BLUE SQUARE THIN LARGE) were presented to the system after each or the foregoing sequences. In the first case the positive part of the model remains unchanged since there are no inconsistencies. The negative example is merely added to the negative part of the model. In the second case, the example contradicts the cluster whose description is (LARGE). This cluster is broken up by a call to REBUILD-CLUSTER using the example (BLUE SQUARE THIN LARGE) as the constraint. The result is two clusters:

CLUSTER
DESCRIPTION

(RED THIN LARGE)
EXAMPLES
(RED CIRCLE THIN LARGE)
(RED SQUARE THIN LARGE)
CLUSTER
DESCRIPTION
(GREEN THICK LARGE TRIANGLE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)

These two clusters replace the old cluster in the positive part of the model:

POSITIVE-PART
CLUSTER
DESCRIPTION
(RED THIN LARGE)
EXAMPLES
(RED CIRCLE THIN LARGE)
(RED SQUARE THIN LARGE)
CLUSTER
DESCRIPTION
(GREEN THICK LARGE TRIANGLE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)
CLUSTER
DESCRIPTION
(RED SQUARE)
EXAMPLES
(RED SQUARE THIN LARGE)
(RED SQUARE THICK SMALL)
CLUSTER
DESCRIPTION
(BLUE TRIANGLE THIN SMALL)

EXAMPLES

(BLUE TRIANGLE THIN SMALL)

NEGATIVE-PART

(BLUE SQUARE THIN LARGE)

(BLUE CIRCLE THIN SMALL)

The other clusters, (RED SQUARE) and (BLUE TRIANGLE THIN SMALL) are consistent with the negative example (BLUE SQUARE THIN LARGE), so they need not be broken up. Note that the negative example has been added to the negative part of the model. The final stage of the model updating process is to check for maximality of the clusters in the positive part.

This process can be described as follows. The current set of clusters form one set, and the answer will ultimately be in a second set. This second is initialized to the empty set. One by one the clusters will be moved from the first to the second set. Before it is added to the second set, the cluster is compared with each of the clusters already in the set. If the two clusters are compatible, that is, if their descriptions can be merged consistently, then the clusters are combined by substituting the merged description for the original descriptions, and the union of the examples lists for the original example lists. The cluster being moved over is combined in this way with every cluster in the answer set with which it is compatible. Only when a cluster is not compatible with ANY of the clusters already there, is it added as an additional cluster to the answer set.

In taking the union of two example sets it is helpful to have some means of identifying equal or redundant examples. This can be done by defining a special function for testing equality. A general approach is to define equivalence of examples in terms of the GREATER-GENERALITY? predicate. A is equivalent to B if and only if both (GREATER-GENERALITY? A B) and (GREATER-GENERALITY? B A) are true. Equivalent examples can then be considered equal for purposes of the set union operator.

Let us illustrate this process with the cluster set above. For simplicity the clusters will be represented merely by their descriptions. The clusters start out on the left, and will be moved to the answer set on the right. The following is the initial state:

CLUSTER <empty-set>

(RED THIN LARGE)

CLUSTER

(GREEN THICK LARGE TRIANGLE)

CLUSTER
(RED SQUARE)
CLUSTER
(BLUE TRIANGLE THIN SMALL)

The first cluster is simply moved across:

CLUSTER
(GREEN THICK LARGE TRIANGLE)
CLUSTER
(RED SQUARE)
CLUSTER
(BLUE TRIANGLE THIN SMALL)

CLUSTER
(RED THIN LARGE)

The second cluster is moved next. It is paired with the cluster on the right, and checked for compatibility. The merged description is (LARGE). This is inconsistent with the first example (BLUE SQUARE THIN LARGE). Note that consistency in this context means consistent with each of negative examples in the negative part of the model. Because of the inconsistency of this merge the clusters are not combined, and the new cluster is added as a second element to the answer set:

CLUSTER
(RED SQUARE)
CLUSTER
(BLUE TRIANGLE THIN SMALL)

CLUSTER
(GREEN THICK LARGE TRIANGLE)
CLUSTER
(RED THIN LARGE)

When the (RED SQUARE) cluster is moved across it is compared with each of the two clusters on the right. When merged with (GREEN THICK LARGE TRIANGLE) it yields the empty description, which is inconsistent with each of the two negative examples. When merged with (RED THIN LARGE) the resulting description is simply (RED). This is consistent with both negative examples, so the two clusters are combined. The union of their respective examples lists is:

(RED CIRCLE THIN LARGE)

(RED SQUARE THIN LARGE)
(RED SQUARE THICK SMALL)

since the example (RED SQUARE THIN LARGE) occurs in both clusters. Thus the new (RED) cluster has these three examples as its examples set. The resulting state of the process is:

CLUSTER
(BLUE TRIANGLE THIN SMALL)

CLUSTER
(GREEN THICK LARGE TRIANGLE)

CLUSTER
(RED)

The final cluster from the left is compared with each of the clusters on the right. The merge of (BLUE TRIANGLE THIN SMALL) with (GREEN THICK LARGE TRIANGLE) yields the description (TRIANGLE). This is consistent with each of the negative examples. The merge of (BLUE TRIANGLE THIN SMALL) with (RED), however, yields the empty description (). This is not consistent with the negative examples.

Thus only the (BLUE TRIANGLE THIN SMALL) and (GREEN THICK LARGE TRIANGLE) clusters are combined. The union of their examples lists is simply:

(GREEN THICK LARGE TRIANGLE)
(BLUE TRIANGLE THIN SMALL)

since there are no redundant examples in this case.

The final state is:

<empty-set>

CLUSTER
(TRIANGLE)

CLUSTER
(RED)

At every stage the clusters in the answer set (on the right) are maximal. Thus in the final answer, which incorporates each of the original clusters, the clusters are all maximal. Nothing was done in this process to violate consistency, either, so the resultant set of clusters is both maximal and consistent.

These clusters are substituted for the positive part in the final model:

"RED-OR-TRIANGLE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(RED)
EXAMPLES
(RED CIRCLE THIN LARGE)
(RED SQUARE THIN LARGE)
(RED SQUARE THICK SMALL)
CLUSTER
DESCRIPTION
(TRIANGLE)
EXAMPLES
(GREEN THICK LARGE TRIANGLE)
(BLUE TRIANGLE THIN SMALL)
NEGATIVE-PART
(BLUE SQUARE THIN LARGE)
(BLUE CIRCLE THIN SMALL)

"either a red thing or a triangle"

This model is the same as the one arrived at by the first teaching sequence presented in section 4.2. augmented by the one additional example. Both models have the same cluster descriptions and will therefore cause the system to recognize the same set of examples as being "red-or-triangle" using either model. In this case the models were made to converge by just one additional example.

4.4 Using networks as descriptions

Feature sets are not the only description scheme that can be used with this theory. As mentioned in section 3.1, any scheme that satisfies the basic assumptions:

monotinicity of concepts
existence of generality predicate
existence of merge operation

can be employed. As a further illustration these notions will be worked out for some examples using networks as the descriptions.

4.4.1 Network descriptions

A network will be described as a set of variables (nodes) and a set of assertions (n-ary links) about these variables. For a naive first treatment, AKO links will not be treated any differently than other links or assertions. A Winston arch example might be described as follows:

```
((NODES (A B C))
 (LINKS (AKO A BRICK)
        (AKO B BRICK)
        (AKO C BRICK)
        (SUPPORTS A C)
        (SUPPORTS B C))) where any atom not specified as a node is assumed to be a
```

known constant (such as BRICK, AKO, and SUPPORTS).

4.4.2 GREATER-GENERALITY? predicate

One way of supplying a greater-generality? predicate is by means of a subgraph isomorphism. That is, description A is more general than description B if and only if A is a subgraph of B under some suitable identification of nodes. Observe that for graphs the subgraph relation is directly analogous to the subset relation for sets.

One simple implementation of this operator has been tested out. A brute force matcher tries out all possible node identifications and checks to see whether the assertion sets satisfy the subset relation (where assertions are considered the same if they match term for term, considering nodes to match if they are paired in the specific identification selected).

Thus the description

```
((NODES (A B))
 (LINKS (AKO A BRICK)
        (AKO B WEDGE)))
```

would be considered of greater generality than

```
((NODES (X Y Z))
```

(LINKS (SUPPORTS Y X)
(AKO X WEDGE)
(AKO Y BRICK)))

because under substitution of the node identifications (A Y) and (B X), the assertions of the first description become (AKO Y BRICK) and (AKO X WEDGE), both of which are contained in the assertion set of the second description.

The following points should be noted. It is not necessary for a node to have any AKO assertion about it at all, since for the present AKO is being treated just like any other assertion or feature. In identifying the nodes of the two descriptions, it is sufficient that the nodes of the first description be paired with some subset of the nodes of the second, not necessarily all of them. Of course the brute force identifier must consider all possible identifications with all possible subsets. The procedure for substituting nodes has to be smart about detecting possible variable conflicts and avoiding these, or on the other hand the system can be restricted by requiring that the node names of different descriptions will always be distinct. Finally, the brute force approach to matching is cumbersome and inefficient, and certainly heuristic approaches can be tried that might provide vast improvement, but efficiency of matching is not the issue being addressed here.

4.4.3 MERGE operator

For feature sets, set intersection was employed as the MERGE operator. For networks the MERGE operator can be implemented as network intersection, that is, the finding of maximal common subgraphs. As with the GREATER-GENERALITY? predicate, the procedure first considers all pairings of nodes, substitutes accordingly for the nodes in the first set, then intersects the two assertion sets (sets of links). The largest intersection set is the winner (since it is guaranteed to be maximal both numerically and relatively). The node set is then pruned so as to include only those nodes that actually appear among the assertions in the intersection. This node set and the intersection set of assertions make up the new MERGEd description. This description is guaranteed to be more general than each of the descriptions which contributed to the merge. In the case of greater-generality the procedure could terminate as soon as one identification of nodes was found that led to containment of the assertion sets. With the merge, it is necessary to search all the identifications, at least until an intersection is found that has size equal to the minimum of the two separate assertion sets.

Applying the above procedure to the descriptions of a "tower"

```
((NODES (A B C))
 (LINKS (AKO A BRICK)
        (AKO B BRICK)
        (AKO C BRICK)
        (SUPPORTS A B)
        (SUPPORTS B C)))
```

and an "arch"

```
((NODES (X Y Z))
 (LINKS (AKO X BRICK)
        (AKO Y BRICK)
        (AKO Z BRICK)
        (SUPPORTS X Z)
        (SUPPORTS Y Z)))
```

yields a description including four assertions:

```
((NODES (X Y Z))
 (LINKS (AKO X BRICK)
        (AKO Y BRICK)
        (AKO Z BRICK)
        (SUPPORTS X Z)))
```

which was obtained by the node identifications (A X)(B Z)(C Y). There were some other possible merges with the same size of the resultant assertion set. These arise from alternative pairings of nodes. So one is chosen arbitrarily.

The MERGE operator is not unique, and it is not symmetric. Its result may depend on idiosyncracies of the order of nodes or assertions. But it is guaranteed to always provide a maximal network intersection that is more general than each of its arguments, which is all the disjunctive learning theory requires.

4.4.4 The monotonicity restriction on concepts

The monotonicity condition requires that for all concepts, any specialization of a positive example will also be a positive instance. This excludes cases where the lack of a feature is critical, that is, the presence of the feature would make the description a negative example. Such a case

occurs in the Winston arch where the lack of the TOUCHING feature for the two support bricks is critical (but the feature is not explicitly included in the description of the positive example). Possible ways to handle this kind of situation will be discussed in section 8.2.

4.4.5 Illustration with disjunctive ARCH in Blocks World

Winston's system learned that an ARCH could have either a brick or a wedge as a crosspiece. The only reason this was possible was that the system already knew about the concept PRISM, which was a common generalization of brick and wedge. The disjunction was therefore implicit in this PRISM concept. Without this device Winston's system would fail on such an example.

The disjunctive learning system proposed here does handle such cases. Instructed with (LEARN ARCH <ARCH1> POSITIVE), where <ARCH1> is the decription

```
((NODES (X Y Z))
 (LINKS (AKO X BRICK)
        (AKO Y BRICK)
        (AKO Z BRICK)
        (SUPPORTS X Z)
        (SUPPORTS Y Z))),
```

the system constructs a simple model with just one cluster:

```
ARCH
POSITIVE-PART
        CLUSTER
                DESCRIPTION
                        <ARCH1>
                EXAMPLES
                        <ARCH1>
NEGATIVE-PART
```

With a second example (LEARN ARCH <ARCH2> POSITIVE), where <ARCH2> is the same description except with a wedge for the supported crosspiece,

```
((NODES (A B C))
 (LINKS (AKO A BRICK)
        (AKO B BRICK)
        (AKO C WEDGE)
        (SUPPORTS A C)
        (SUPPORTS B C))),
```

the system merges the new example into the one cluster resulting in the overgeneralization:

```
ARCH
    POSITIVE-PART
        CLUSTER
            DESCRIPTION
                <DESCR1>
            EXAMPLES
                <ARCH1>
                <ARCH2>
    NEGATIVE-PART
```

where <DESCR1> is the description

```
((NODES (A B C))
 (LINKS (AKO A BRICK)
        (AKO B BRICK)
        (SUPPORTS A C)
        (SUPPORTS B C))).
```

The AKO feature for C has been dropped. Now the system will judge as instances of ARCH anything regardless of what c is, as long as the other features are satisfied. Suppose that there are somethings like that that are not intended to be arches, such as if c was a pyramid, a foo, a person, or a theory. Then even though bricks A and B supported C the feature (AKO C PYRAMID) or foo, person, etc., would make the example a noninstance.

The instruction (LEARN ARCH <NON-ARCH> NEGATIVE) is given, where

<NON-ARCH> is the description

((NODES (A B C))
(LINKS (AKO A BRICK)
(AKO B BRICK)
(AKO C PYRAMID)
(SUPPORTS A C)
(SUPPORTS B C))).

Since this negative example satisfies the description <DESCRI> of the cluster, the cluster must be broken up. This results in the disjunctive model:

ARCH
POSITIVE-PART
CLUSTER
DESCRIPTION
<ARCH1>
EXAMPLES
<ARCH1>
CLUSTER
DESCRIPTION
<ARCH2>
EXAMPLES
<ARCH2>
NEGATIVE-PART
<NON-ARCH>

"<ARCH1> or <ARCH2>"

The system does not at present have the ability to form partial disjunctions. Disjunctions are so far either all or nothing. A facility for partial disjunctions would be an interesting and useful extension.

4.4.6 Illutration with UNCLE-NESS concept in Kinship

Relations Domain

The concept of UNCLE-ness is also disjunctive. This provides yet another illustration of networks used for descriptions within the disjunctive learning theory.

Given the instruction (LEARN UNCLE <UNCLE-1> POSITIVE) where <UNCLE-1> is

```
((NODES (A B C))
 (LINKS (AKO A PERSON)
        (AKO B PERSON)
        (AKO C PERSON)
        (MALE C)
        (B PARENT-OF A)
        (C BROTHER-OF B)))
```

the system creates the straightforward model with one cluster.

Given the second instruction (LEARN UNCLE <UNCLE-2> POSITIVE) where <UNCLE-2> is

```
((NODES (A B C D))
 (LINKS (AKO A PERSON)
        (AKO B PERSON)
        (AKO C PERSON)
        (AKO D PERSON)
        (MALE D)
        (B PARENT-OF A)
        (C SISTER-OF B)
        (D HUSBAND-OF C)))
```

the system merges the two descriptions to form the model:

```
UNCLE
    POSITIVE-PART
        CLUSTER
            DESCRIPTION
```

<DESCR-1>
EXAMPLES
<UNCLE-1>
<UNCLE-2>
NEGATIVE-PART

where <DESCR-1> is the description

```
((NODES (A B C))
 (LINKS (AKO A PERSON)
        (AKO B PERSON)
        (AKO C PERSON)
        (B PARENT-OF A)
        (MALE C))).
```

The concept has been overgeneralized to the description "3 persons, one a parent of the second, and the third a male".

A counterexample to this is then supplied by the instruction

(LEARN UNCLE <NON-UNCLE> NEGATIVE)

where <NON-UNCLE> stands for the description

```
((NODES (A B C))
 (LINKS (AKO A PERSON)
        (AKO B PERSON)
        (AKO C PERSON)
        (MALE C)
        (B PARENT-OF A)
        (C PARENT-OF B)))
```

The system discovers that this negative example satisfies the description of the cluster in the old model. Thus the old model's cluster must be split up, yielding the new model:

UNCLE
  POSITIVE-PART
    CLUSTER
      DESCRIPTION
        <UNCLE-1>
      EXAMPLES
        <UNCLE-1>
    CLUSTER
      DESCRIPTION
        <UNCLE-2>
      EXAMPLES
        <UNCLE-2>
  NEGATIVE-PART
    <NON-UNCLE>

"either <UNCLE-1> or <UNCLE-2>"

Of course the system also has the capability to eliminate extraneous features through the merging process. If the above descriptions were more complicated, for example, including extraneous features such as age, genders of persons for which their gender is not relevant to the concept, proper names, etc., the system could still acquire the concept. It would naturally require additional examples to eventually eliminate the extraneous features from each of the terms of the disjunction.

5.0 SYMMETRIC TREATMENT OF POSITIVE AND NEGATIVE EXAMPLES

5.1 Outline of the approach

Certain concepts are more compactly expressible as negations of disjunctions than as direct disjunctions. An example is:

"anything that is not either blue or a circle". To express this concept as a direct disjunction could lead to a possibly infinite number of terms, since there is no limit to the descriptions that might lack blue-ness and circle-ness.

This situation suggests that it might be desirable to simultaneously build a model of the negation of a concept while developing the model of the concept itself. This parallel model constructing could be completely symmetric with respect to the handling of positive and negative examples. If it was observed that the negative model was compact and efficient while the positive model was cumbersome, then greater weight could be given to the simpler negative model.

In the original theory, the positive-part of a model was structured as a cluster-set while the negative-part was simply a set of examples. In the symmetric version of the theory, both the positive-part and the negative-part will be structured as cluster sets. Implicit in each of these cluster sets is an example set, which can be recovered by taking the union of the examples from each of the clusters in the set. The intention is that the clusters will satisfy a symmetric or double maximality and consistency condition. The clusters of the positive-part will be maximal and consistent with respect to the examples in the negative part. Similarly, the clusters in the negative-part will be maximal and consistent with the examples of the positive part. To guarantee that these conditions are maintained, it is necessary to modify the operations that update the model.

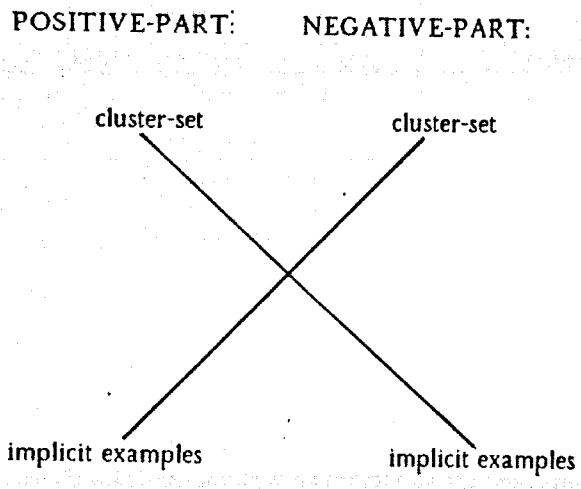
Before, there were two basic operations performed when updating a model:

1. INSERTION = Adding a new example into a cluster list (subject to the constraints of the known negative examples).
2. SPLITTING = Using a new negative example to break up any positive clusters which had become overgeneralized.

Operation 1 was used when a positive example was encountered, and operation 2 when there was a negative example.

In the symmetric version of the theory both operations 1 and 2 will be performed for

every example, positive or negative. For a positive example, operation 1 is applied to add the example to the positive-part cluster set subject to the consistency constraint with the examples implicit in the negative-part, and operation 2 is used to break up over-general clusters in the negative-part cluster set. Symmetrically, for negative examples, operation 1 is used to add the example to the negative cluster set relative to the positive examples, and operation 2 uses the new negative example to break up any over-general clusters in the positive-part.



In this diagram the lines indicate the constraint relationships for consistency and maximality.

In order to be certain that the symmetric version of the system will work properly, it is necessary to strengthen the monotonicity assumption of section 3.1. Previously, it was only assumed for positive examples that all descriptions of lesser generality would also be positive examples. Now it is necessary in addition to assume the analog for negative examples: that any examples of lesser generality (that is, having all the same features plus some additional ones) will also be negative examples.

5.2 Illustrations of the symmetric theory in attributes world (using feature sets as descriptions)

Suppose the concept to be learned is "neither a circle nor a square". The concept can be taught using the following sequence of examples:

1. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (RED CIRCLE THIN) NEGATIVE)
2. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (RED TRIANGLE THICK SMALL) POSITIVE)
3. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (BLUE TRIANGLE THICK LARGE) POSITIVE)
4. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (RED SQUARE LARGE THIN) NEGATIVE)
5. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (RED RECTANGLE THIN SMALL) POSITIVE)
6. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (BLUE CIRCLE LARGE) NEGATIVE)
7. (LEARN "NEITHER-CIRCLE-NOR-SQUARE" (GREEN THICK SQUARE SMALL) NEGATIVE)

Example 1 "NEITHER-CIRCLE-NOR-SQUARE" (RED CIRCLE THIN) NEGATIVE

The system builds a model with a single cluster in the negative-part. Note that the system, just like the asymmetric version, can make use of negative examples even when they occur at the beginning of the teaching sequence.

The two basic operations of INSERTION and SPLITTING are trivial in this case, since at the start both parts are empty.

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART

NEGATIVE-PART
CLUSTER
DESCRIPTION
(RED CIRCLE THIN)
EXAMPLES
(RED CIRCLE THIN)

Example 2 "NEITHER-CIRCLE-NOR-SQUARE" (RED TRIANGLE THICK SMALL) POSITIVE

With this the system creates a single cluster in the positive-part of the model:

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(RED TRIANGLE THICK SMALL)
EXAMPLES
(RED TRIANGLE THICK SMALL)
NEGATIVE-PART
CLUSTER
DESCRIPTION
(RED CIRCLE THIN)
EXAMPLES
(RED CIRCLE THIN)

Again the operations proceed in a very straightforward manner. The insertion is done relative to the negative constraint of the single negative example (RED CIRCLE THIN). The splitting operation simply checks that the new positive example does not contradict any of the negative clusters (the trivial single cluster in this case).

Example 3 "NEITHER-CIRCLE-NOR-SQUARE" (BLUE TRIANGLE THICK LARGE) POSITIVE

The insertion operator adds the new example to the positive-part, relative to the constraint of the negative example (RED CIRCLE THIN). This leads to a new cluster with description (TRIANGLE THICK), which is consistent with the negative example.

The splitting operation again leads to no action.

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART
CLUSTER

DESCRIPTION
(TRIANGLE THICK)
EXAMPLES
(BLUE TRIANGLE THICK LARGE)
(RED TRIANGLE THICK SMALL)
NEGATIVE-PART
CLUSTER
DESCRIPTION
(RED CIRCLE THIN)
EXAMPLES
(RED CIRCLE THIN)

Example 4 "NEITHER-CIRCLE-NOR-SQUARE" (RED SQUARE LARGE THIN) NEGATIVE

The insertion operation adds the new negative example to the negative-part of the model subject to the constraint of the two positive examples (BLUE TRIANGLE THICK LARGE) and (RED TRIANGLE THICK SMALL). The example merges into the cluster to form the description (RED THIN), which is consistent with the two examples given as constraints.

The splitting operator checks the clusters (one in this case) of the positive-part to see if consistency is violated. Since the cluster description (TRIANGLE THICK) is not satisfied by the new negative example, no splitting is necessary.

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(TRIANGLE THICK)
EXAMPLES
(BLUE TRIANGLE THICK LARGE)
(RED TRIANGLE THICK SMALL)
NEGATIVE-PART
CLUSTER

DESCRIPTION
(RED THIN)
EXAMPLES
(RED SQUARE LARGE THIN)
(RED CIRCLE THIN)

Example 5 "NEITHER-CIRCLE-NOR-SQUARE" (RED RECTANGLE THIN SMALL) POSITIVE

Since the example is positive, it must be added to the positive-part of the model. Merging with the cluster description (TRIANGLE THICK) would yield the empty description (), which is contradicted by the negative constraints (RED SQUARE LARGE THIN) and (RED CIRCLE THIN). Therefore this new example goes to start a new cluster.

The splitting operator notes that the negative cluster (RED THIN) is satisfied (and therefore contradicted) by the new positive example. Thus this cluster needs to be split up, using the new example as the consistency constraint. This yields the model:

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(TRIANGLE THICK)
EXAMPLES
(BLUE TRIANGLE THICK LARGE)
(RED TRIANGLE THICK SMALL)
CLUSTER
DESCRIPTION
(RED RECTANGLE THIN SMALL)
EXAMPLES
(RED RECTANGLE THIN SMALL)
NEGATIVE-PART
CLUSTER

DESCRIPTION
(RED SQUARE LARGE THIN)
EXAMPLES
(RED SQUARE LARGE THIN)
CLUSTER
DESCRIPTION
(RED CIRCLE THIN)
EXAMPLES
(RED CIRCLE THIN)

Example 6 "NEITHER-CIRCLE-NOR-SQUARE" (BLUE CIRCLE LARGE) NEGATIVE

This negative example is added to the second cluster yielding the description (CIRCLE), but it is not added to the first since the merged description would be (LARGE), which is contradicted by the positive example (BLUE TRIANGLE THICK LARGE).

The splitting operator has no effect here since the positive clusters are both consistent with the new example.

The new model is:

"NEITHER-CIRCLE-NOR-SQUARE"
POSITIVE-PART
CLUSTER
DESCRIPTION
(TRIANGLE THICK)
EXAMPLES
(BLUE TRIANGLE THICK LARGE)
(RED TRIANGLE THICK SMALL)
CLUSTER
DESCRIPTION
(RED RECTANGLE THIN SMALL)
EXAMPLES

(RED RECTANGLE THIN SMALL)

NEGATIVE-PART

CLUSTER

DESCRIPTION

(RED SQUARE LARGE THIN)

EXAMPLES

(RED SQUARE LARGE THIN)

CLUSTER

DESCRIPTION

(CIRCLE)

EXAMPLES

(BLUE CIRCLE LARGE)

(RED CIRCLE THIN)

Example 7 "NEITHER-CIRCLE-NOR-SQUARE" (GREEN THICK SQUARE SMALL) NEGATIVE

The new negative example is here added to the first cluster of the negative-part of the model but not the second. The resulting description for the first cluster is simply (SQUARE).

There are no splittings of clusters in the positive part since both are consistent with the new negative example.

The final model after the above teaching sequence:

"NEITHER-CIRCLE-NOR-SQUARE"

POSITIVE-PART

CLUSTER

DESCRIPTION

(TRIANGLE THICK)

EXAMPLES

(BLUE TRIANGLE THICK LARGE)

(RED TRIANGLE THICK SMALL)

CLUSTER
DESCRIPTION
(RED RECTANGLE THIN SMALL)
EXAMPLES
(RED RECTANGLE THIN SMALL)
NEGATIVE-PART
CLUSTER
DESCRIPTION
(SQUARE)
EXAMPLES
(GREEN THICK SQUARE SMALL)
(RED SQUARE LARGE THIN)
CLUSTER
DESCRIPTION
(CIRCLE)
EXAMPLES
(BLUE CIRCLE LARGE)
(RED CIRCLE THIN)

positive: "either a thick triangle or a small thin red rectangle"
negative: "neither a square nor a circle"

There are now two viewpoints for the concept model, a positive view and a negative one. Either viewpoint (or both) could be disjunctive in character. The above illustrative teaching sequence led to a model where both the positive and negative parts were disjunctive. It would be easy to implement various schemes for selecting one of the viewpoints as being simpler or more economical than the other. One such scheme would favor the smaller number of terms (clusters) in the disjunction, and in case of ties choose the one with the smaller number of total features summed over all terms. This scheme, in particular, would select the negative point of view for the above concept example. For purposes of identification, an example would then be judged a positive instance only if it did not satisfy any of the terms (clusters) of the negative-part of the model.

Note also how the operations of insertion and splitting worked to always maintain mutual maximality and consistency of positive clusters with negative examples and negative clusters with

positive examples.

6.0 A LIMITED MEMORY VERSION OF THE THEORY

The system as described implicitly remembers all the examples (both positive and negative) ever presented to it for each concept it has learned. This assumption is perhaps a little unrealistic. Although people do show a memory for certain examples presented, they certainly do not have perfect memory.

It would be interesting to explore alternative schemes with weaker memory requirements. This is additionally motivated by the fact that as the number of examples goes up, so does the combinatorics of the manipulation employed in updating the model.

One proposal is to have the system remember only concepts that lead to a change in the model. There are two informal arguments in favor of this. First, examples leading to contradictions of expectations are, from a psychological perspective, more likely to be noticed, and more processing time will be devoted to them, hence they might have a greater likelihood of being remembered. From the viewpoint of the system, counterexamples are more likely to be useful in the future, while examples which are consistent with the model are more likely to be redundant with other examples, and therefore, less worthy of being remembered explicitly.

Such a scheme has the distinct advantage that the size of the representation of a concept will tend to stabilize as the model of the concept converges. Under the original memory assumption, the model would continue to grow, even after the model was in "perfect agreement" with the given concept. Sometimes, though, the system using the limited memory scheme would reject potentially relevant examples, simply because they were redundant with the "wrong" model. Then the system would have to be given a similar example later, or be reminded of the particular "forgotten" example. This seems a small price to pay for the great overall reduction of memory requirements.

7.0 CURRENT STATE OF THE IMPLEMENTATION

The framework learning system described in chapter 2 has been fully implemented, including the feedback mechanism, whereby the system learns from the teacher's feedback regarding correctness of response to identification requests. All programming was done using the language LISP.

Both the asymmetric and symmetric versions of this theory of learning disjunctive concepts have been programmed and tested. A decision procedure has not been implemented for selecting a positive or negative viewpoint for the symmetric model, though this is not for reasons of difficulty.

Both the basic and the symmetric versions of the disjunctive theory have been tested using (1) feature sets and (2) networks (as described in chapters 4 and 5) as the representational (or descriptive) scheme. The brute force approach used for the network merging was very inefficient for more complicated examples.

The limited memory version of the theory described in chapter 6 has not been implemented at this point. It should not be a difficult project, and the results of testing it and comparing it to the other system should be interesting.

8.0 CRITICISM OF THE THEORY

8.1 Advantages

The primary strengths of the proposed theory are that it allows recovery from overgeneralization and that it enables the learning system to acquire disjunctive as well as the more usual conjunctive concepts.

The system is relatively immune to variations in the order of the teaching sequence. Although such variations do exert some influence on the form of the model, they never cause the system to get stuck in a nonrecoverable way. If two different sequences for teaching the same concept lead to different models, then by presenting appropriate additional examples, the two models can be made to converge to equivalent models.

The ability to treat positive and negative examples in a symmetric manner seems useful and appealing. Further exploration is necessary to determine if this is, in fact, a good idea.

8.2 Disadvantages

One disadvantage of this system over previous systems, is its somewhat heavy reliance on memory of the specific examples presented. In chapter 6 a proposal was made for reducing this dependence on memory, which leads to a somewhat more plausible memory requirement. Nevertheless, this proposal is not yet implemented, and it is uncertain just how effective the modified theory would be.

Another criticism concerns efficiency of computation. Some of the operations of model updating, in particular the operations for restructuring cluster lists by breaking up clusters, and for guaranteeing maximality of clusters, require a fair number of MERGE operations. The MERGE operator is likely to be the most expensive of all the basic operations, especially in the case of more complex descriptive schemes such as networks or frames. When checking maximality there are O(n**2) MERGES where n is the number of clusters (or terms of the disjunction). For the restructuring operation the requirement for MERGE operations is roughly O(n) in the number of examples so far remembered.

Still another limitation of the theory is its reliance on the monotonicity assumptions about the concept space. In some cases the system still works despite violations of this condition, but the condition is essential to any attempt to prove the correctness of the system's behavior. In particular, one of the examples employed by Winston does not conform to the monotonicity criterion. This is the famous ARCH example where the arch description:

```
((NODES (A B C))
 (LINKS (BRICK A)
        (BRICK B)
        (BRICK C)
        (SUPPORTS A C)
        (SUPPORTS B C)))
```

is a positive example and the more specialized description:

```
((NODES (A B C))
 (LINKS (BRICK A)
        (BRICK B)
        (BRICK C)
        (SUPPORTS A C)
        (SUPPORTS B C)
        (TOUCHES A B)))
```

fails to be positive because of the additional feature (touches a b). One way of viewing this problem is to assume there was an "implicit" (does-not-touch a b) feature in the first description. Then the monotonicity condition would not have been violated. The result of presenting the negative example is then to bring into relief such implicit features, so the system can then treat them just as any other feature. This line of thought suggests a way of incorporating certain of Winston's ideas within the framework of the current theory for learning disjunctive concepts.

Another problem, which I consider to be a major one, shared by this sytem and many previous ones, is that the system assumes that the description of an example presented to it will include every one of the features potentially relevant to learning that concept. A truly flexible learning system must be able to invent new features, redescribe examples in novel ways, and decide which classes of features are relevant to learning a given concept, and which should be ignored. The case of the "hidden" features described in the last paragraph is just one special case of this general phenomenon. In Winston's descriptions (of arches for example) the fact that certain of the bricks were supported by the table might be relevant to learning a concept different from arch. I'm not suggesting that it is possible to include every possible feature in advance, (probably it is not possible), but I believe that it is important for a learning system to have the capability to go back to an example or scene and reexamine it for new features that it might have overlooked or simply not included in the initial description. This issue becomes clearer in systems that are themselves responsible for creating descriptions of scenes which are given to them. It would then be nice for

the system to go back for a second look at a scene to create a new or modified description for purposes of refining the concept model. Such an action would certainly be necessary if the system ended up with the same description for a positive scene and a negative one. This is another challenging problem for further work.

The final limitation to be listed here is that the system has no capability for forming partial disjunctions. Sometimes all the descriptions will correspond (and be merge-able) except for one feature. In such cases it might be more reasonable to localize the disjunction to the single feature where the variation takes place. The system as it stands would form two completely separate descriptions such as

(A and B and C) or (A and B and D)

rather than the slightly more economical

(A and B and (C or D)).

Another examples is that of the Brick and Wedge ARCHES presented in section 4.4.5.
Certainly the representation of partial disjunctions leads to a more complex representation scheme, but it also may provide sufficient additional flexibility and economy as to justify the trade-off. A compromise solution could be the induction of auxiliary disjunctive concepts such as the PRISM concept that Winston's system had within its AKO hierarchy.

8.3 Discussion of related work

Systems such as Winston[18] and Hayes-Roth[6] are essentially generalizing programs. They move only in the direction of greater generality. Winston's program actually did have a limited ability to recover from certain overgeneralizations by backtracking to choice points and following an alternative branch. Nevertheless, the overall flavor is one of pure generalization. Hayes-Roth's system is a particularly pure example of the "generalization only" approach to concept learning. He employed a network representation scheme which he called "parameterized structural descriptions". Given a set of examples, his system employed heuristic network matching to find a maximal common subgraph. This description corresponds to the most conservative conjunctive generalization of the set of examples.

Neither of these systems had the capability of learning disjunctive concepts. Winston's system did include a facility for representing disjunctive concepts, but it could not form its own disjunctions, thus it could never learn new disjunctive concepts. Hayes-Roth's system did not even include a mechanism for representing disjunctive concepts. It dealt exclusively with conjunctive

descriptions.

One of the similarites of the present work with that of Winston [18], is the important role played by negative examples in each. In Winston's ARCH learning program, negative examples were crucial to introducing emphatic, or "MUST-BE" pointers, into the concept model. In the present work, negative examples play a dual role, inhibiting overgeneralizations and initiating disjunctive splitting. Hayes-Roth's induction system [6] made no use of negative examples.

One very important difference between the present system and previous systems such as those of Winston [18] and Hayes-Roth [6], is with respect to memory requirements. Winston and Hayes-Roth maintain only a single concept model after any sequence of examples. The examples themselves are forgotten. My system, on the other hand, remembers the examples which led to the current concept model. These examples were necessary in order to enable recovery from overgeneralizations. Chapter 6 proposed a method for reducing the extent of this memory burden, but it was still necessary to remember some of the examples, though the number of these was greatly reduced as the model converged. I conjecture that some memory of examples is necessary in order to enable the learning of disjunctive concepts. Perhaps this is the price that must be paid to enable recovery from overgeneralizations in situations where disjunctive concepts are a possibility.

Another interesting induction system is that of Vere [17]. He proposes a mechanism called "Counterfactuals". Concepts are described by a general rule modified by a description of the exceptions. Vere extends this to "multilevel counterfactuals" where there may be exceptions to the exceptions, and so forth. Such a mechanism has an appealing flavor of "homing in" or converging to a correct model. Discovering and enumerating exceptions is one very important way of recovering from overgeneralizations. In some situations it is also clearly the most efficient method. For example, in learning the formation of the past tense of verbs, it is efficient to represent a general rule of adding the "-ed" ending, and then explicitly list the irregular verbs that are exceptions to this rule.

An important way in which the system of counterfactuals proposed by Vere differs from the other systems mentioned is in the way it is taught. The previous systems all learn from a sequence of examples which are presented one at a time. Each new example leads to an updated model of the concept. Vere's system assumes that all of the examples are presented simultaneously as a set. The model building process is not incremental. If an additional example is later encountered that does not conform to the induced model, it is necessary to start over completely to generate a new model using all the previous examples as well as the single new example.

Vere's COUNTERFACTUAL system is nevertheless quite interesting in that it does deal

with disjunctive concepts. Vere's approach is analogous with the minimization of Boolean functions as a minimal sum of products. A single conjunctive description corresponds to a product term in a sum of products expression. A disjunctive description can be thought of as the sum of its conjunctive terms. This system for the representation of disjunctive concepts is very similar to that employed in the present work, where the clusters are the structures analogous to the terms of the disjunction. The important difference lies in the fact that my approach is incremental whereas Vere's is not. My system, like Winston's and Hayes-Roth's, builds a model which can be incrementally updated on the basis of subsequent individual examples.

Sherman and Ernst [15] present a very different approach to the problems of concept learning in general and the learning of disjunctive concepts in particular. Their work is an extension of the EPAM discrimination net learning model proposed by Feigenbaum [3] in the context of learning conjunctive concepts such as strings of letters. The approach is one of concept differentiation or discrimination, as opposed to generalization. What is seen as important is not the defining of a concept in isolation, but discriminating between a given concept and other learned concepts.

The memory of their system is organized as a discrimination net with tests at each node. The net is actually a binary tree. The test associated with each node, when applied to a presented example, decides which branch of the tree should be followed, and thus which further tests should be applied. A given input example is classified by starting at the top of the tree, applying the test associated with that node, selecting a subtree according to the result, and recursively classifying the input according to the subtree thus selected. This process leads to a unique terminal node selected by the tests applied. The terminal nodes of the tree have category labels instead of tests. Thus, any input can be classified into one of the categories labelled at a terminal node of the tree. This is the basic recognition process.

Note that more than one label may exist at a single terminal node. This may occur either when the concepts corresponding to the labels have not yet been discriminated, or when there is more than one name for a single concept. Also, a single label may reside at more than one terminal node. This feature allows the representation of disjunctive concepts. Each terminal node corresponds to a conjunctive description, formed from the results of the tests applied on the path leading to it. By repeating a single category label at different terminal nodes, it is possible to achieve the effect of a disjunctive sum of conjunctive terms.

Learning proceeds by adding new tests and branches to the tree, in order to permit discrimination of a larger number of concepts. When told that it has incorrectly classified a given

input, the system creates a new test to discriminate between the new input and the former concept category. This approach is very appealing in certain respects. It eliminates the necessity of having a complete description of each example immediately upon its initial presentation. Rather, the description of a concept can be built up slowly as more and more features are discovered to be necessary to differentiate it from other concepts. Such an approach also has important psychological plausibility. People often start out with crude or vague definitions of concepts. The definitions become more precise as it becomes necessary to make finer and finer discriminations.

One of the drawbacks of this approach is that the model of a concept is not collected in a single place, but rather is distributed throughout the network in terms of the various tests. This makes it more difficult for the system to make generalizations. It is unclear how to observe similarities between different descriptions. The common features may be distributed along different paths in different branches of the tree.

Yet another restriction is that the tests must be applied in a fixed order. Thus if one test could not be applied due to insufficient information being available, the system could not proceed to recognize it at all. It is desirable for a recognition system to degrade more gracefully than this. Humans seem able to recognize things even when a few of the features are missing or obscured.

Due to the strengths of this approach, further investigation is warranted. In particular, the representation system is accompanied by a natural recognition procedure, learning does not presuppose complete descriptions of examples, concepts are learned in relation to one another, and disjunctive concepts are no more difficult to learn than simple conjunctive ones.

8.4 Directions for future work

There are numerous directions in which further exploration is needed. One important area is that of composite or compound concepts. This is the study of how concepts combine to form more complex concepts. Such a compounding mechanism would also contribute to the learning of disjunctive concepts by allowing the localization of disjunctions. If two descriptions matched exactly except for a few features, they could be merged by means of an auxiliary disjunctive concept that collected the differing features. In this way the common features can be exploited, and the disjunction can be confined or localized, rather than extending to the complete descriptions. Such a mechanism would also permit the nesting of disjunctions, which has not been addressed in this work.

Another important issue is that of the discovery and invention of new descriptive

features. It is unrealistic to assume that all the relevant features of a concept are present in the initial description given to a learning system. An important aspect of learning is the invention of new ways of describing things, using new features, or other newly acquired concepts.

The organization of concepts in the memory store presents still more problems for further inquiry. How should concept models be stored so as to economize on space by exploiting similarities and common structures? How can the memory organization support efficient retrieval and recognition processes?

How can the learning system be designed so as to generate its own auxiliary or internal concepts without the prodding of an external teacher? That is, how can an intelligent system be made to recognize important or regularly occurring patterns, and initiate its own concept generalization and concept composition processes, in the course of its normal activity?

Bibliography

1. Brown, J.S., "Steps Toward Automatic Theory Formation", Proc. 3rd IJCAI, Stanford, Calif., 1973, pp. 121-129.

2. Bruner, J.S., The Process of Education, Cambridge: Harvard University Press, 1960.

3. Feigenbaumm, E., "The Simulation of Verbal Learning Behavior", in Feigenbaum, E., and Feldman, J., eds., Computers and Thought, New York: McGraw-Hill, 1963, pp. 297-309.

4. Fikes, R., Hart, P., and Nilsson, N., "Learning and Executing Generalized Robot Plans", Technical Note 70, SRI, Menlo Park, Calif., July 1972.

5. Hayes-Roth, F., "Patterns of Induction and Associated Knowledge Acquisition Algorithms", Dept. of Computer Science, Carnegie-Mellon University, May 13, 1976.

6. Hayes-Roth, F., and McDermott, J., "Knowledge Acquisition from Structural Descriptions", Working Paper, Dept. of Computer Science, Carnegie-Mellon University, Feb. 11, 1976.

7. Jones, T.L., A Computer Model of Simple Forms of Learning, Ph.D. Thesis, MIT, (Project MAC Technical Memorandum 20), January, 1971.

8. McMaster, I., A Proposal for Computer Acquisition of Natural Language, TR-75-3, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, May 1975.

9. Michalski, R.,and Negri, P., "An Experiment on Inductive Learning in Chess Endgames", in Elcock, E., and Michie, D., eds., Machine Intelligence 8, Halstead Press, New York, 1977, pp.175-192.

10. Miller, P.L., "An Adaptive Natural Language System that Listens, Asks, and Learns", Proc. 4th International Joint Conference on Artificial Intelligence held at Tbilisi, Georgia, USSR, 1975, vol.1, pp.406-413.

11. Minsky, M., "A framework for representing knowledge", AI Memo 306, MIT Artificial

Intelligence Laboratory, 1974.

12. Negri, P., "Inductive Learning in a Hierarchical Model for Representing Knowledge in Chess End Games", in Elcock, E., and Michie, D., eds., Machine Intelligence 8, Halstead Press, New York, 1977, pp.193-204.

13. Rumelhart, D., and Norman, D., "Accretion, Tuning, and Restructuring: Three Modes of Learning", Technical Report 63, Center for Human Information Processing, Univ. of Calif., San Diego, August, 1976.

14. Salveter, S., "Learning Structures to Represent Verb Meaning", Computer Sciences Technical Report #294, Computer Sciences Deptartment, University of Wisconin-Madison, March 1977.

15. Sherman, R., and Ernst, G., "Learning Patterns in Terms of Other Patterns", Pattern Recognition, vol.1 (1969), pp.301-313.

16. Smith, R., Mitchell, T., Chestek, R., and Buchanan, B., "A Model for Learning Systems", Stanford Heuristic Programming Project, Memo HPP-77-14, Computer Science Department, Stanford University, March 1977.

17. Vere, S., "Multilevel Counterfactuals for Generalizations of Relational Concepts and Productions", Dept. of Information Engineering, University of Illinois at Chicago Circle, 1978.

18. Winston, P., Learning Structural Descriptions from Examples, Ph.D. Thesis, MIT, AI TR-231, 1970.

19. Winston, P., "Learning By Creating and Justifying Transfer Frames", AI memo AIM-414a, AI Laboratory, M.I.T., Cambridge, Mass., January, 1978.

20. Winston, P., "Learning by Understanding Analogies", AI Memo AIM-520, AI Laboratory, M.I.T., Cambridge, Mass., June 1979.