A MODEL FOR ADAPTIVE PROBLEM SOLVING APPLIED TO NATURAL LANGUAGE ACQUISITION

Larry R. Harris, Ph.D.

TR 72-133

Math Dept (Bradley Hall)
Dartmouth College
Hanover, N.H. 03755

June 1972

Department of Computer Science
Cornell University
Ithaca, New York 14850

A MODEL FOR ADAPTIVE PROBLEM SOLVING APPLIED TO NATURAL LANGUAGE ACQUISITION

Presented to the Faculty of the Graduate School
of Cornell University for the Degree of
Doctor of Philosophy

by
Larry Ronald Harris

August 1972

BIOGRAPHICAL SKETCH

Larry R. Harris was born in St. Louis, Missouri on October 19, 1947. In 1969 he received a Bachelor of Science degree from Cornell University. He began graduate study at Cornell in the Department of Computer Science in September 1969. He is a member of Tau Beta Pi, and the Association for Computing Machinery. The author is married and has two children.

to my Family

ACKNOWLEDGEMENT

The author is indebted to many people for their assistance in the preparation of this thesis. I would like to thank my special committee, Professors Henry David Block, chairman and thesis advisor, Ellis Horowitz, and William Lucas.

One's ideas on a subject go back to many sources and it would be impossible to list all of them here. But I would like to express my gratitude for the particularly stimulating discussions during the past year with Professors Terry Winograd, Robert Constable, Alan Shaw, Ira Pohl, John Dennis, John Holland, David Gries, and Jerome Feldman. The author found the book _Problem Solving Methods in Artificial Intelligence_ by Nils Nilsson to be an excellent source of fruitful ideas.

I would like to once again thank Professors Block and Horowitz for their suggestions, criticism, and overall assistance throughout the preparation of this paper.

ABSTRACT

A MODEL FOR ADAPTIVE PROBLEM SOLVING APPLIED TO NATURAL LANGUAGE ACQUISITION

Larry R. Harris, Ph.D.

Cornell University, 1972

Adaptive Problem Solving is the application of artificial intelligence learning techniques to practical problems. The approach taken in studying Adaptive Problem Solving is three-fold. First, to develop a model for Adaptive Problem Solving in order to specify the processes involved in computer learning, as well as the interaction between these processes. Second, theoretically well-founded, practical algorithms are developed for each of these learning processes. Third, as an application of this theory, the Natural Language Acquisition Problem is formulated in terms of the adaptive model.

The specification of algorithms to perform the learning processes leads to the development of the Bandwidth Heuristic Search, an extension of the heuristic search, that includes many practical considerations without forfeiting any theoretical capabilities. A modification of this algorithm, the Bandwidth Heuristic Search for MIN/MAX trees, is shown to be superior to the alpha-beta minimax process.

The model is applied to the Natural Language Acquisition Problem in order to force an encounter with several critical problems involved with computer learning. The Natural Language Acquisition Problem is the problem of providing a robot the adaptive mechanisms sufficient to learn to converse with a human teacher using natural language. The robot first learns the lexicon of the language by correlating the teacher's description of the robot's actions with the robot's internal description. Then the robot infers a grammar that reflects the structure of the teacher's sentences. At this point the robot can begin conversing using a natural language. The linguistic capability of the robot includes the ability to disambiguate lexical and structural ambiguities, and the ability to formulate full sentence replies. After several learning sessions the robot converses in English using nested dependent clauses.

This adaptive linguistic system successfully copes with many of the critical problems involved in computer learning and serves as an example of an adaptive program in which the learning, rather than yielding only minor improvements, provides the primary basis for successful performance.

TABLE OF CONTENTS

~~The computer has become a tool for problem~~ solving for a wide range of applications. The number of obstacles encountered in the solution of some problems seems to challenge human capability to enumerate them. Thus, we see the advent of computer programs that

Rewrite words

1. Introduction detect and cope with these obstacles by themselves. Such an adaptive approach to problem solving implies the notion of improving performance with time and thus have been called "learning" programs. Using the computer to derive results to practical problem in this manner defines the field of Adaptive Problem Solving.

The approach we take in studying adaptive problem solving is three-fold. First to organize the state-of-the-art knowledge of computer learning into one general model. Second, for each of the learning processes identified by the model we will suggest theoretically well-founded algorithms to perform them. Third, as a test of the model, we will apply it to a challenging problem, the Natural Language Acquisition Problem.

In order to specify the processes involved in computer learning we develop a model for adaptive problem solving. Examining important past research in computer learning in light of this model identifies the critical problems involved in adaptation related to the interaction of these processes. The virtue of the model is its ability to clearly specify these processes and their interaction for a wide variety of problem domains.

We use the adaptive model as a guide to study the algorithms for performing these learning processes. In our analysis of algorithms we will insist on both a firm theoretical basis and practical efficiency for the algorithms we select. For some processes such algorithms already exist, for others we will develop them. In particular, one such algorithm developed for this purpose is the Bandwidth Heuristic Search, an extension of the Heuristic Search (Nilsson(1971)), that encorporates many additional practical considerations into its theoretical development.

After using the model to define the learning processes and the algorithms to perform them, we will apply the model to an interesting application. In order to force an encounter with some of the critical problems involved in computer learning, we apply the adaptive model to the Natural Language Acquisition Problem. An adaptive approach to the use of natural language by computer has many desirable advantages over the conventional approach. A learning linguistic program is not restricted to a particular vocabulary, or a particular type of sentence, or even to a particular language. The result is a language-independent system more readily extendable in terms of both lexicon and sentence structure than a straightforward programmed approach.

This particular application of the adaptive model provides an example of a learning program in which the learning is a first order effect. Successful game playing programs that employ learning have been criticized because the optimization of parameter weights is of secondary importance compared to the development of the parameters in the first place. The adaptive linguistic system improves its performance, not by adjusting weights, but by altering its entire view of the structure of the language. The complexity of the language used by the adaptive system, which starts with no a priori knowledge of this language, eventually reaches that of nested dependent clauses in English.

The adaptive approach also allows treatment of certain more complex linguistic capabilities, such as the formulation of concepts and speaking, that do not often receive treatment in computer linguistic systems.

For these reasons we feel the Natural Language Acquisition Problem serves as a successful application of the adaptive model, demonstrating the effectiveness of the suggested algorithms, and the overall generality of the model in specifying a complex problem.

2. A Model for Adaptive Problem Solving

2.1 Overview of the Model

In order to specify the problem-independent processes involved in computer learning we formulate a model for adaptive problem solving. These processes are:

1) The Objective Function - a success criterion.

2) The Adaptive Routine - a strategy generator.

3) The Strategy Testing Routines - a mechanism for testing strategies.

```
                                        ┌-------------------------------┐
                                        |                               |
                                        |  STRATEGY TESTING ROUTINES    |
                                        |                               |
┌----------┐   ┌-----------┐            |  ┌-----------------------┐    |
|          |   |           |            |  |   HEURISTIC SEARCH    |    |
| ADAPTIVE |-->| OBJECTIVE |            |  └-----------------------┘    |
|          |   |           |====>       |        |                      |
| ROUTINE  |<--| FUNCTION  |            |  ┌---V-------┐  ┌---------┐   |
|          |   |           |            |  |EVALUATION|  |         |   |
└----------┘   └-----------┘            |  |    OF    |->| MEMORY  |   |
                                        |  |HEURISTIC |  |         |   |
                                        |  └----------┘  └---------┘   |
                                        └-------------------------------┘
```

A strategy is a specific means of solving the prescribed problem. The representation of a strategy is not restricted by this model, as will become apparent when we discuss a variety of problem domains.

The meaning of "heuristic" is restricted in this model to the notion of estimated cost as used in the heuristic search (see Section 2.5.2.1). Therefore the process labeled

"Evaluation of Heuristic" means the <u>calculation</u> of a well defined function, not the rating of a specific "heuristic" trick.

Learning, as formulated by this adaptive model, comes about as the adaptive routine suggests better and better strategies to solve the problem. The relative value of each strategy is determined by the objective function, which may require the use of the strategy testing routines in making this determination. Based on this measurement of old strategies, the adaptive routine tries to suggest increasingly improved strategies as time goes on. In this way the performance of the system improves with time.

The advantage gained by this formulation of an adaptive system is that one can take advantage of theoretically well-founded algorithms to perform the learning processes. As we shall see in a detailed discussion of each process, these effective algorithms offer distinct advantages over ad hoc methods so often applied in computer learning systems.

2.2 Examples - How Important Past Research Fits the Model

In this section we discuss various important past research in adaptive systems in light of the adaptive model. We hope to indicate the generality of the model by showing its ability to represent these widely different problem domains. Our purpose in this section is to build a familiarity with the model and the accompanying notation by describing familiar examples within the framework. In Section 2.6, after analyzing the model in detail, we will return to these examples to gain insight on the advantages of specifying problems within this adaptive framework. The strong points, as well as the weak points, of each of these adaptive systems will be made more apparent by our analysis of the model.

Samuel's checker program (Samuel(1957,1962))

Probably the most successful adaptive program to date is Samuel's checker program. The program plays by using an alpha-beta minimax routine to search for the best next move. A static evaluation function is used at the bottom of the move tree to rate the board positions. For this purpose a relatively fixed set of detectors which measure aspects of board configurations, such as piece advantage and center control, are used. The output of these detectors is a positive or negative number which indicates the extent to which the program or the opponent has the advantage with respect to each criterion. The manner in which these board parameters are combined to calculate the static evaluation distinguishes his earler system (Samuel(1957)) from his more recent one (Samuel(1962)).

In the initial system a linear polynomial evaluation scheme is used to calculate the static evaluation. If (p(1) ... p(k)) are the board parameter detector values and (c(1) ... c(k)) are the weights to indicate the importance of the parameters then the summation of the products c(i)*p(i) constitutes the evaluation function.

In his later version Samuel introduced a signature table evaluation scheme in order to consider non-linear relations between board parameters. Each table is a tabulated function over groups of quantized detector values (called a signature type). The output of each table is a quantized numeric value indicating the "value" of this particular combination of board parameters. The output of the first level tables are fed into second level tables and so on. The output of the final table is the evaluation function.

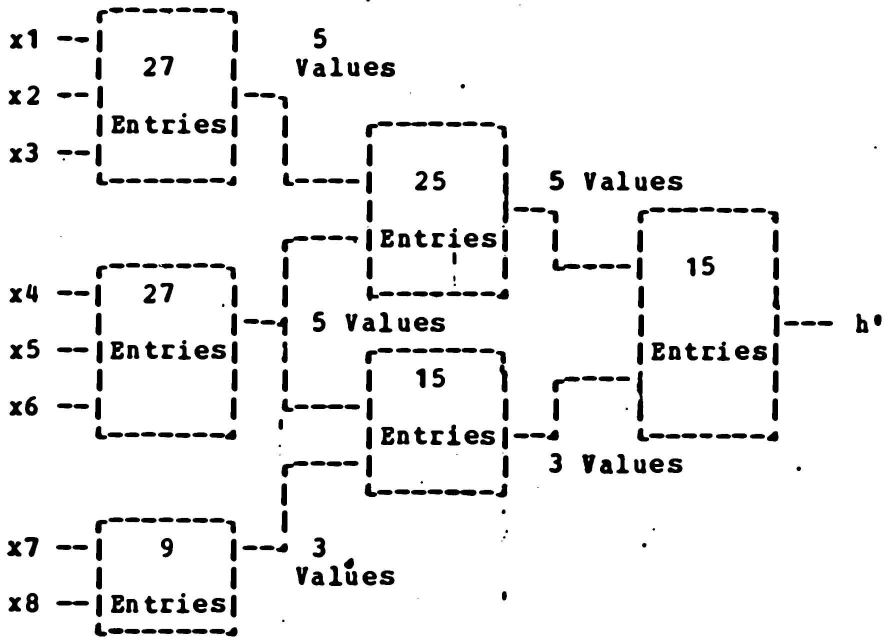

FIGURE 2.2a

Since the goal of Samuel's work was to study machine learning, he avoided programming known strategies into the system. The program was to develop its own playing strategy by either playing actual games or being told "book" moves made by human checker masters. In the initial work the playing strategy was changed by changing the weights (c(1) ... c(k)). For the later version the playing strategy was changed by varying the entries (the tabular function values) of the signature tables. In either case the play of the program changes because the static evaluation is changed and thus the minimax may backup a different prefered move.

Since the program learns from both competitive playing, called "generalization learning", and studying book moves, called "book learning", we shall describe Samuel's adaptive systems with respect to both of these types of learning. The signature table program works only with book learning.

"GENERALIZATION LEARNING"

strategy - the vector of weights (c(1) ... c(k)) used in the linear polynomial evaluation.

objective function - called DELTA, the difference between the static evaluation of the current position and the backup value calculated by the minimax look ahead.

adaptive routine - change weights by an Operant Reinforcement technique (see Section 3.3.2). for changing weight c at step n+1 c(n+1) = (1-1/m)*c(n) + (1/m)*Z, where Z is the positive or negative stimulus, and m starts at 16 and moves up in powers of 2 until it reaches 256.

strategy testing routines - alpha-beta minimax program to calculate DELTA.

evaluation of heuristic - by the linear polynomial scheme.

memory - storage of board positions and parameter values to speed (and thus improve) the look ahead process.

"BOOK LEARNING"

Book learning was done with both the linear polynomial and signature table methods. Since the adaptation proceeds in an analgous manner for each method, we will represent only the signature table method.

strategy - quantized entries of the signature tables.

objective function - calculate A & D, the number of times in a test of N book moves a signature type causes the static evaluation to be (A) above, or (D) below the book move.

adaptive routine - change entries in the signature tables to the value (A-D) / (A+D).

strategy testing routines - calculates the best move from the "book" situation using static evaluation and minimax technique.

evaluation of heuristic - by signature tables.

memory - none.

Klopf's pattern recognition program

Klopf's program is an adaptive pattern recognition system. The structure of Klopf's system is similar to that of Samuel's that both use a weight-detector scheme.

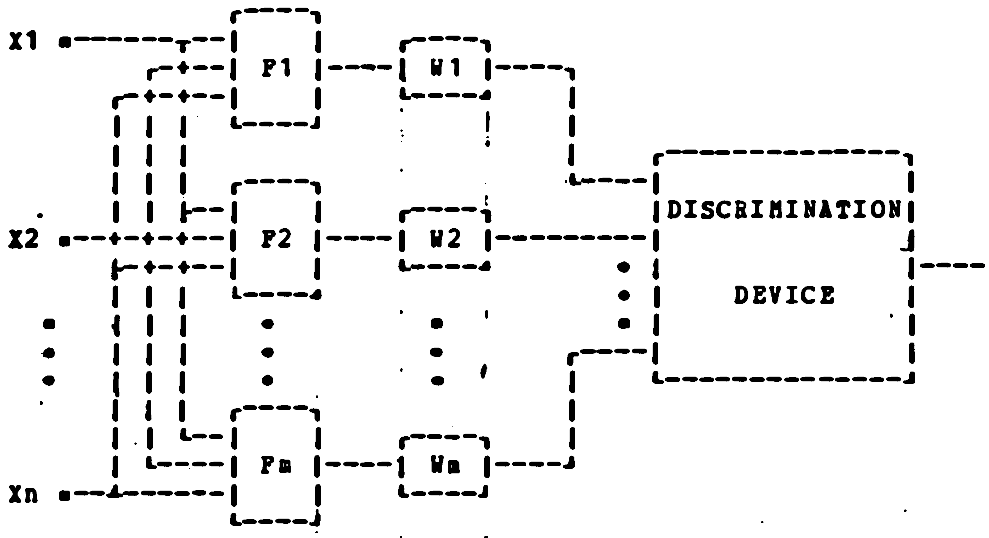

FIGURE 2.2b

However, the adaptive approach is just the opposite of Samuel's. Klopf adjusts the detectors not the weights. He uses an optimal discrimination algorithm to calculate the weights given a set of m detectors. In this case the detectors are arbitrary functions of n binary inputs. The adaptation takes place by replacing functions with low associated weights by new functions.

<u>strategy</u> - a set of m arbitrary functions of n binary inputs.

<u>objective function</u> - a function proportional to the percentage of patterns correctly classified and inversely proportional to the number of patterns incorrectly classified.

<u>adaptive routine</u> - replace poor functions (those with low weights) by a new randomly generated function.

<u>strategy testing routines</u> - calculate optimal weights for the current set of functions, then apply a set of test patterns.

<u>evaluation of heuristic and memory</u> - none.

Fogel, Owens, and Walsh evolutionary system

Fogel et al have written an adaptive system to find a finite state machine to predict the next input symbol after receiving a particular input sequence. The technique is to start with a given finite state machine and keep making random changes in the state transition table. They keep repeating this process until the performance of the machine stops improving.

strategy - a finite state machine

objective function - the percentage of symbols correctly predicted by the finite state machine.

adaptive routine - randomly change an element of the state transition table (i.e. change the output or the next state of a transition).

strategy testing routine - run the current finite state machine on the input sequence.

evaluation of heuristic and memory - none.

2.3 Objective Function

The objective function is a function whose domain is the set of strategies and whose range is the set of real numbers. In its simplest sense the objective function is simply an interface between the adaptive routine and the strategy testing programs. It performs the bookkeeping involved in testing a given strategy in a number of trial situations. As we allow the notion of this interface to evolve into a well specified function, it begins to take on greater and greater importance. By studying the form of many typical objective functions we can better choose an appropriate adaptive routine. It will also become clear that the objective function controls the "type" of learning the system does. By this I mean we can perform either "generalization learning" or "book learning" within this model simply by changing the objective function and keeping everything else constant.

Finally, in our analysis of the objective function we will consider a degenerate case, called "telephone pole space." Unfortunately this type of function arises often in artificial intelligence work, and thus, it is important that we consider it in some detail.

2.3.1 Formal Definition

The purpose of the objective function is to rate the strategies on their ability to perform some desired task. The objective function rates the strategies by using them in some test situation. The simplest type of objective function for a game playing situation might be:

$F(s)$ = the percentage of games won by using strategy s in a test of N games against various opponents.

Although this type of objective function is much too coarse for practical problems, it does fit the required form for an objective function. That is, given any strategy it returns a number that indicates the relative merit of the strategy. This simple objective function is implemented by performing the bookkeeping of a number of calls to the strategy testing programs. With this simple example in mind let us develop a formal definition of the objective function.

We want the objective function to reflect the inherent value of the strategy s compared to other strategies. Therefore $F(s1) > F(s2)$ if s1 is a "better" strategy than s2. The type of function described thus far is a "rank order". It simply ranks the strâtegies in order of their value. It is necessary to extract more information than this about a strategy from the objective function. We would like to think that if $F(s1) = 2*F(s2)$ then s1 is somehow twice as good as s2. This type of function is a "ratio scale". Therefore we will say that the objective function defines a ratio scale over the strategy space.

For some problems we can define the objective function F as a continuous function over the strategy space. For other problems the objective function may not be so structurally simple. The idea of closeness or "connection" between two strategies becomes less meaningful as we consider non-numeric strategies. Minsky(1961) warns that for many problems we must contend with this structural difficulty.

> "... we need some additional structure on the search space. This structure need not bear much resemblance to the ordinary spatial notion of direction, or that of distance, but it must somehow tie together points which are heuristically related. We will call such a structure a heuristic connection. We introduce the term for informal use only -- that is why our definition is itself so informal. But we need it. Many publications have been marred by the misuse, for this purpose, of precise mathematical terms, e.g., metric and topological."

This notion of "heuristic connection" between strategies will be more explicitly defined in Section 2.4.6.

2.3.1.1 Sample Objective Functions

The above objective function for game playing (F1) was said to be too coarse for practical purposes, because we would like to extract more information from playing an entire game with a strategy than just the fact that it won or lost. Consider the following two examples of objective functions for game playing.

F2(s) = the percentage of "book" moves duplicated by a given strategy in a trial set of N book moves.

F3(s) - inversely related to the sum of the differences between the evaluation of board positions in a trial set, and the back-up evaluations of the trial set resulting from look-ahead.

Each of these objective functions depends only on trials of one move instead of full games. In this case we can have very large trial sets since the cost of a trial is so much less than playing an entire game.

Consider the resulting difference in using these various objective functions for a given game. They all try to find a winning strategy in the long run, but go about it in very different ways.

F1 goes straight for the desired result by viewing a strategy as good only if it wins. The problem with this approach is that it is extremely costly to play an entire game and we are getting very little information from each game. In particular, it would be good to know which particular feature of the strategy caused it to win or to lose.

F2 tacitly makes the assumption that if a strategy can mimic enough "book" moves then it will win. A "book" move is one made by a master human player. This objective function would produce a more rapid improvement in play than F1 since we are gaining more information about the tested strategy per cost of testing the strategy. The only flaw is that the strategy may be making the right move for the wrong reasons. But this "noise" should average out over a large trial set.

F3 tries to find a strategy that will enable a deeper look-ahead in the move tree. If a strategy is good in predicting how things will be further down the move tree, then we can search the move tree to a greater depth, and thus play better. The problem here is that it may not be possible for a strategy of the form we are using to accurately predict these future values.

It should be clear that the use of these various objective functions will yield very different styles of play, in that their intermediate objective is different. Thus we can say that the objective function controls the learning process.

2.3.2 Control on "types" of Learning

It is an important feature of this model that various objective functions can be used with the same adaptive routine, and the same strategy testing routines. The result is that the model can encompass various types of learning by changing only the objective function. The model using F2 performs "book learning"; with F1 & F3 it performs "generalization learning". Making use of mass memory devices with any of these objective functions we can perform "rote learning". Thus, we see the inherent difference between these various types of learning is implicitly contained within the objective function and the use of memory.

2.3.3 Form of Common Objective Functions

Since we will call upon the adaptive routine to find the maximum of the objective function we should investigate the form of some common objective functions. We will consider F1, F2, & F3 as well as another function F4, an objective function for grammatical inference. In this case a strategy is a grammar and we are given a set of sentences, called the "positive set", that the desired grammar should generate. Another set of sentences, the "negative set", should not be generated by the grammar. Using these terms we can define an objective function to measure the worth of an individual grammar.

<u>F4(s)</u> = the percentage of positive sentences parsed by s less the percentage of negative sentences parsed by s.

Let us consider the form of these objective functions.

(1) <u>no closed form</u> - Since these functions are based on simulations no simple expression of the function can be found.

(2) <u>stochastic</u> - Playing an entire game with a strategy (as in F1) may have some randomness involved in determining which player moves first. For most games this can seriously bias the results of the game and thus the value of the objective function. This means that we cannot depend heavily on the objective function since it may actually return different values for the same strategy at different times. Of course, if the number of games in the trial set is large, this "noise" will average out.

(3) have a high variance - Since the objective function F2 uses a test set which is a sample from many possible situations, the resulting function value will vary from the "true" value, the "true" value being defined as the percentage of book moves duplicated by the strategy in all possible board situations. Since we cannot afford to test a strategy under all these situations, we must sample from this set as is done using F2. This sampling process introduces some variance that is inversely related to the size of the trial set.

(4) multi-modal - The objective function F4 will have many strategies that are rated positively. This is due to the fact that there are infinitely many grammars to produce a finite number of positive sentences. The fact that many strategies may perform well also arises in game playing. In this domain it is not unusual for many strategies to perform better than their surrounding strategies. The result is a local maximum in the objective function.

(5) flat - Objective functions F1 & F4 will produce extremely low values for many strategies in a give region. There will be many game playing strategies that will not win any games in the trial set and thus rate a low value. There are many grammars that will not be able to parse even one positive sentence, and thus rate a low value.

With this picture of the functions we are trying to optimize we can consider the algorithms for doing so. First we should consider a degenerate case of the objective function called "telephone pole space".

2.3.4 Telephone Pole Space

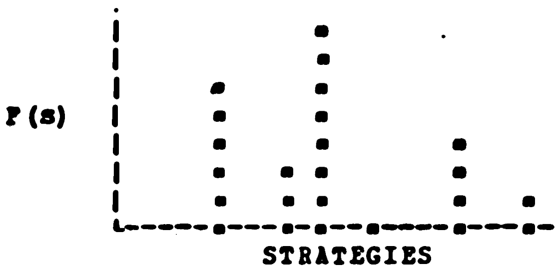

FIGURE 2.3.4

A telephone pole space has low values for virtually all strategies. For a sparse few strategies the objective function exhibits a large positive value. This is the extreme case of a function being both multi-modal and flat. Sampling the strategy space tells us nothing about the location of the maximum. We are in no better position to predict a good strategy after sampling than we were before sampling.

An example will help dramatize the effect of a telephone pole space. Adapting on a continuous objective function is a lot like the Hot and Cold game played by children. It is somewhat easier than this game because we receive more information than just "good" or "bad". We are told by the objective function proportionally how "hot" or "cold" the strategy is. (Since the objective function is a ratio scale and not just a rank order).

Adapting on a telephone pole space is to search without

..ing told anything about your closeness to the maximum until ... actually select it. This means we could be arbitrarily ...se to the global maximum and not even know it.

That dealing with a telephone pole space is a critical ...blem in artificial intelligence work is made evident by ...sky(1961):

> "It is often supposed that this false-peak problem is the chief obstacle to machine learning by this method. This certainly can be troublesome. But for really difficult problems, it seems to us that usually the more fundamental problem lies in finding any significant peak at all. Unfortunately the known E functions (objective functions in our terminology -- LFH) for difficult problems often exhibit what we have called (Minsky and Selfridge, 1960) the "mesa phenomenon" in which a small change in a parameter usually leads to either no change in performance or to a large change in performance. The space is thus composed primarily of flat regions or "mesas." Any tendency of the trial generator to make small steps then results in much aimless wandering without compensating information gains."

It is clear that in dealing with a telephone pole space ..r a "mesa space" either corrective action must be taken or a ..ndom adaptive search might just as well be used. Of course, the random search would be impractical for virtually all ...oblems.

The corrective action taken for the telephone pole space ..countered in the natural language acquisition problem will be to define operators to move about the telephone pole space from telephone pole to telephone pole. In this manner we avoid sampling all the bad strategies. This approach may not be possible for other instances of telephone pole objective functions, but it indicates the type of corrective action that is absolutely required in order to obtain robust adaptation on this type of objective function.

2.3.5 Higher Order Objective Functions

We have considered thus far objective functions that return only a single numeric value. Objective functions of this type are called first order functions (Holland(1969)). It may be desirable to feed back more information about a strategy than just a single numeric value. If this is the case, then we call this a higher order objective function.

Examples of higher order objective functions can be found in both game playing and grammatical inference. Samuel's checker player feeds back additional information on each board parameter. This information is used to determine whether the associated weight of the parameter should be increased or decreased. Higher order feedback may occur in grammatical inference if we wished to pass back statistics on the use of each of the productions during the test. The adaptive routine could then use this information to help decide which productions are to be deleted or changed.

The line between first order and higher objective functions is a fine one. We cannot simply look to see if the objective function returns a single number, because it may be possible to encode a great deal of extra feedback into a single number. For many problems extra feedback of the type we have mentioned can be useful in aiding the adaptive routine to make improvements in the strategy. However, it is not clear that higher order systems are computationally more powerful than first order systems (Holland(1961)).

2.4 Adaptive Routine

The purpose of the adaptive routine is to search the strategy space for a maximum of the objective function. The adaptive routine will suggest new strategies on the basis of information gained by sampling old strategies.

A great deal of work has been done on numerical optimization procedures, but for reasons that will be made apparent soon, few of these techniques can be applied to artificial intelligence problems. For this reason some researchers have used ad hoc adaptive routines for specific applications. Some of these ad hoc routines have been shown to be less effective than a random search of the strategy space (Minsky(1961), Lindsay(1968)). I believe we have reached the point in formalizing algorithms for artificial intelligence problems, that the use of ad hoc routines is no longer necessary. After an analysis of the adaptive process within the artificial intelligence environment we will suggest certain algorithms that are both theoretically and practically effective. Thus, we feel it is no longer necessary to develop ad hoc techniques given the existence of these well developed algorithms.

Given the form of the objective function developed in the previous chapter, we will consider a number of classes of adaptive routines and analyze how they perform on these functions.

In this chapter we will also consider the problems of adapting on non-numeric functions and adapting on telephone pole spaces, two important problems which arise often within artificial intelligence domains and in particular we must contend with them when we consider the Natural Language Acquisition Problem in Section 3.

2.4.1 Formal Definition

In the cases where the objective function is a continuous function over the strategies, the adaptive routine is simply an unconstrained optimization algorithm. In these cases the adaptive routine uses the objective function values of old strategies to calculate a new strategy. Generally the estimated slope, or the gradient of the objective function is used to predict a new strategy.

For non-numeric problems and telephone pole spaces, this approach cannot be applied. We must develop different techniques, possibly making use of second order feedback, in order to cope with these special problems.

We will therefore define the adaptive routine to be any process that transforms old strategies into new ones, in whatever representation is used, using the value of the objective function and second order feedback. We will rate adaptive routines by their ability to suggest improved strategies. We will consider a random search of the space to have zero value as an adaptive routine, even though it may suggest an improving sequence of strategies. Using this scale we note that some past researchers (Freidberg(1958,1959)) have actually used ad hoc routines that would rate negatively since they were less effective than a purely random search of the strategy space (Minsky(1961)).

2.4.2 Constraints Imposed by the Form of the Objective Function

In the last chapter we considered the form of some typical objective functions. Since the adaptive routine is to find the maximum of of a function of these types, we must consider the form of the objective function in choosing the appropriate adaptive routine.

1) no closed form - Since no closed form of the objective function exists we cannot symbolically differentiate the function, set the derivative to zero and solve for the maximum. We will encounter many optimization techniques that try to numerically approximate the derivative, or more generally, the gradient. These "gradient techniques" will then calculate a new strategy by moving along the gradient, or hill-climbing. We will keep these "gradient techniques" in mind as we consider further the form of the objective function.

2) & 3) stochastic and/or high variance - These factors both tend to degrade the objective function's ability to accurately measure the worth of a strategy. For this reason it is unwise for an adaptive strategy to compare two strategies that are close together. This is true because the difference beween the "true" value of the strategy and the function value may be large: because of the stochastic nature

of the function, or the fact that we are testing the strategy on a small trial set. If we compare points far apart the maximum error in estimating the slope between points is diminished.

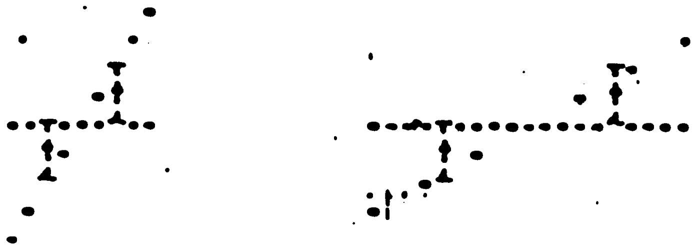

FIGURE 2.4.2

Calculating the slope between two strategies that are close together can lead to grossly inaccurate estimates of the true slope as the above diagram indicates. For this reason the gradient techniques cannot effectively search this type of strategy space, since they often compare strategies a "differential" apart. Empirical results indicate that these "gradient techniques" tend to wander aimlessly about the strategy space, being consistently misled by the inconsistencies in the values of the objective function.

We will be interested in adaptive routines that only compare points that are far apart, to minimize the error in the resulting calculations.

4) multi-modal - The functions we are trying to maximize tend to have many false peaks. It is a major concern to find

the global maximum not just a local maximum. Any adaptive technique that is concerned with only one point at a time is extremely vulnerable to finding false peaks. Because of this fact, many of the routines implement a random perturbation when they have converged to a result. Then the process is started again in the hope that it will again converge to the same result (giving us more confidence that the result is a global maximum).

But there exist other methods that deal with more than one point at a time. These techniques are much less sensitive to false peaks because they can compare various peaks at the same time. These "parallel" techniques also allow for a smattering of initial guesses that allow us to start with many different strategies. For both of these reasons we will prefer the parallel techniques over the sequential ones.

5) <u>flat</u> - Since there tend to be large flat areas in the objective function, we should look for routines with a variable step size. In this manner we can avoid wasting evaluations on this flat region.

2.4.3 Example - Regular Simplex Method

As an example of an adaptive routine that satisfies the constraints mentioned above, let us consider the Regular Simplex Method (Meier(1969)).

Despite the name this technique has nothing to do with the Simplex method of solving systems of linear inequalities. The Regular Simplex Method evaluates the objective function at the vertices and the centroid of an n dimensional regular simplex, where n is the dimensionality of the domain of the objective function. Examples of simplexes are an equilateral triangle being projected onto a function of 2 variables, and a regular tetrahedron on a function of 3 variables. We move about the function domain by reflecting the simplex about one of its sides (faces), if the projection of the simplex has sufficient "tilt". We converge to a maximum by shrinking the simplex, if the simplex is reasonably "flat". This method, unlike other variations of simplex optimization techniques, always retains the regular simplex. It does not attempt to optimize along the reflected direction since this would involve comparing strategies that are close together.

The advantages of this method are: (1) it can cover the function domain rapidly since it has n+2 initial guesses and the initial size of the simplex can be made large; (2) it is insensitive to initial guesses and false peaks since it can be considering strategies far apart at the same time; (3) an added benefit built into this routine is that it can control the variance of the objective function. When comparing points whose values are far apart we do not need as much accuracy in evaluating the objective function, as when we compare points close together. Thus the Regular Simplex Method can call for the objective function to use a larger test set in order to lower the variance and get a more accurate rating of the strategy. This can save a great deal of computer time in the early phases of the adaptation when the strategies are far apart because the simplex is large.

The disadvantages of this technique are: (1) it works only on functions with numeric domains; (2) it moves slowly when the dimensionality of the domain of the objective function is more than 10; (3) the n+2 initial guesses are forced to be related since they must form a regular simplex.

We conclude that this method is good for numeric functions of less than 10 arguments about which we can make intelligent initial guesses.

2.4.4 Constraints Imposed by Non-numeric Spaces

For some artificial intelligence problems it is often convenient to encode a strategy using non-numeric means. This is true for the Natural Language Acquisition Problem. In this case a strategy is the character string representation of a grammar. In these non-numeric cases our choice of an adaptive routine is severely restricted since most research on optimization techniques has been done on functions defined over vector spaces. We must eliminate from consideration any technique that "calculates" the next strategy since this only makes sense when dealing with numbers.

Thus, we are restricted to adaptive methods that do not attempt to approximate the gradient; that are relatively insensitive to initial guesses, and false peaks; that cover the function domain quickly; and that can possibly operate over non-numeric functions. This eliminates the well known numerical techniques such as the quasi - Newton methods and the Davidon - Fletcher - Powell Method. An optimistic note is that we are not interested in finding very accurate results. In most cases only a rough value for the optimum is all the objective function is capable of measuring anyway.

2.4.5 Example - Simulated Evolution

As an example of an optimization technique that can work on non-numeric functions we shall discuss the simulated evolution techniques of Holland(1967,1969), Cavicchio(1970) and Hollstien(1971). This method of adaptation employs a biased random approach similar to the theory of natural selection that has proved to be so effective in evolution. This method has also done remarkably well on the classical test functions used in numerical analysis. As we shall see it fits our needs particularly well.

A "population" of strategy points is created by initial guesses. Then certain operators are randomly applied to selected members of the population. The application of the operators is controlled by a probability distribution constructed from the objective function values. That is to say, the strategies that rate high are more likely to be chosen to create offspring than those strategies that are rated low by the objective function. The new strategies created by the application of the operators to the selected parent strategies are evaluated by the objective function and put into the new generation. The best half of the old generation is also carried forward to the new generation. The size of the population is variable, but generally in the range of 12 to 20.

This method has actually been proved optimal, in the sense that in the limit this adaptive method will do as well as any other adaptive technique, if certain conditions are maintained at each generation (Holland(1969)). This theoretical result and the method's inherent simplicity make it ideal for our needs.

In most of Holland's work exclusive use of the genetic operators is made. Some of these operators, such as mutation, seem to be necessary to make the method work under general conditions. However, other genetic operators such as crossover and inversion appear to be ill-suited for many problems. The forced usage of the genetic operators seems particularly unwarranted since they may not be closed on the strategy space. As noted in Cavicchio(1970), the coding of the strategy was critical in determining the effectiveness of some of these operators. The strategies must be coded in Grey code, not binary code for some operators to work properly. Another example of this is in the Natural Language Acquisition Problem. In this case a strategy is represented by a character string of letters, arrows and markers. Now if we apply inversion to one of these grammars we clearly may get a resulting character string that is not a legal encoding of a grammar. Thus, the operator inversion is not even closed on the strategy space.

For this reason we will not limit ourselves to the genetic operators. We will allow any operator that transforms a legal strategy into a different legal strategy. It is an advantage of this evolutionary technique that new operators can be developed for different domains. The following operators would be convenient for functions ~~with~~ OVER numeric domains.

mutation - This operator simply returns a legal strategy that has been randomly generated and thus bears no resemblence to members of the current population. This operator although used infrequently, should always be included in the set of operators since it gives the overall system the capability of getting "unhooked" from false peaks.

extension - This operator randomly perturbs all the parameters of the parent strategy. Each parameter may be perturbed by an amount dependent upon the domain of that argument. The maximum perturbation in any direction can be controlled dynamically depending on how fast we want the strategies to move about.

marriage - This operator calculates the numeric average between two parent strategies. The resulting strategy is somewhat like each of the two parents, but may also exhibit improved behavior over either of the parents.

In fact, we can use any of the standard numerical analysis optimization techniques as "operators" for this method. However, we have already ruled these out for reasons mentioned earlier. We therefore will consider this method using only "simple" operators of the type just mentioned. Of course, for special domains, we will allow the use of ad hoc operators to be defined, provided they meet certain basic requirements. Such a requirement would be that the operators be capable of covering the strategy space. If this were not true for a set of operators we would be working under the severe restriction of not being able to generate all possible strategies.

This method has many advantages. It covers the function domain extremely fast since we are allowed so many initial guesses. We can effectively cover the strategy space from the start. It is insensitive to initial guesses since the population can be so spread out. It avoids false peaks since members of the population can be examining many such peaks concurrently. Any 1-point method must iteratively compare peaks whereas this method can be comparing them simultaneously. Thus, we tend to find the best peak. If we do get stuck on a false peak, it is still possible that the mutation operator will send a new member of the population in the area of a higher peak. The comparison of the objective function is not critical since the only use of the objective function is made in building the probability distribution function that biases the selection of strategies to be operated upon. The only critical step based on function values is for the lower half of a population. They will be dropped from the population. This method also works on functions with non-numeric domains. We simply introduce operators that will change one strategy to another within the representation scheme.

The simulated evolution technique has the following disadvantages. It tends to "waste" function evaluations because of the randomness of the mutation operator. But it still does well compared to other methods. It is very hard to get results that are extremely accurate. This is not an important disadvantage for artificial intelligence domains. The method has so many important parameters that it is hard to apply it effectively the first time. Since the cost of adaptation is so high, it would be good to gain experience with the parameters. In Cavicchio's(1970) thesis this experimentation with the parameters was done with the conclusion that the method was relatively insensitive to these parameters given that they were "reasonable".

We conclude that this method is excellent for our needs and a good method of at least generating good initial guesses for more accurate methods. For non-numeric problems it is as flexible as the operators we define.

2.4.6 Constraints Imposed by Telephone Pole Space

In this section we develop a technique for coping with telephone pole spaces. We have stated that no general optimization procedure can do better than a purely random search on this type of function, and thus our approach will not be to directly attack the telephone pole space but to reparametrize the objective function thereby redefining the "heuristic connection" between strategies. The resulting function may now be amenable to hill climbing techniques. Let us consider this method of dealing with telephone pole spaces in light of Minsky's(1961) remarks.

> "a profitable search in such a space requires steps so large that hill-climbing is essentially ruled out. The problem-solver must find other methods; hill-climbing might still be feasible with a different "heuristic connection."

Working within our adaptive framework we can more explicitly define "heuristic connection" so that the researcher can know exactly how to proceed to perform this transformation. We will say that two strategies are "close" or "heuristically connected" if one strategy can be transformed into the other strategy by the application of <u>one</u> operator. The operators are the means by which the adaptive routine suggests new strategies. All adaptive routines use such operators, for the "gradient" techniques the operators are the arithmetic operators, for the Simulated Evolution technique the operators are mutation, extension, marriage, or whatever other operators are defined. The Regular Simplex Method uses operators that "shrink" or "reflect" the simplex.

Using this definition we can change the "heuristic connection" by changing the operators, which is to change the means by which we search the strategy space. If we can define operators that when operating on a telephone pole strategy result in another telephone pole strategy, then we have compressed the telephone poles and can employ the aforementioned adaptive routines.

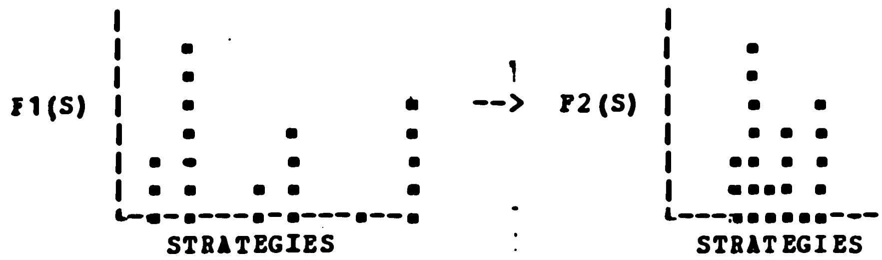

FIGURE 2.4.6

Thus, we have specified an explicit means of coping with a telephone pole space. Of course, we cannot give a general set of operators that will work in all cases because the selection of such operators is a very problem-dependent matter. But because the problem is so critical in computer learning we have chosen to apply the model to the Natural Language Acquisition Problem because it deals with a telephone pole space. In Section 3.4.1 we will specify operators that perform this compression for the grammatical inference involved in natural language acquisition. We offer this application as empirical evidence that telephone pole spaces can be transformed into workable functions, and that the means for performing this transformation fits directly into the adaptive model. As we shall see in Section 2.6, failure to consider this problem has caused some researchers to use adaptive techniques that are only marginally better than a random search.

2.5 Strategy Testing Routines

The strategy testing section is the most problem-dependent section of the model. For many applications the section is simply a trial run of the defined problem. But for a wide class of problems the testing of a strategy will require a tree or graph search.

For example, in a game playing situation we play with a strategy by looking ahead in the move tree. In the natural language domain we can parse a sentence by looking for it in the tree defined by the grammar.

For most artificial intelligence problems we cannot afford to search the entire tree, and thus, we are forced to consider an heuristic search. In this chapter we present the Heuristic Ordered Search (Hart(1968), Slagle(1970), Nilsson(1971)), and discuss the theoretical foundations of the algorithm. For many applications we may not be satisfied by this particular theoretical approach. For example in checkers we would be happy to find any accessible winning position, not just the closest one; if the search would be correspondingly cheaper. We will reorient the theoretical analysis of the heuristic search to accommodate such practical considerations. The result is a new type of heuristic search called the Bandwidth Heuristic Search.

We will also develop a Bandwidth Heuristic Search that can be applied to MIN/MAX trees that is based on these same theoretical foundations. We will compare this new search technique to the alpha-beta minimax procedure, and its extensions.

Finally we discuss the means by which the heuristic is calculated, which has important ramifications throughout the model since it determines the dimensionality of the strategy space.

2.5.1 Simple Testing of Strategies by Simulation

The strategy testing routines perform the actual work required in the testing of a given strategy. The work done by the strategy testing routines is always specified by the definition of the problem. That is to say that in a game playing environment, the strategy testing programs test a strategy by playing under the rules of the game. For the language acquisition problem a strategy (a grammar) is tested by parsing with it. The strategy testing routines carry out the actual playing of the game, or the parsing of the sentences.

Thus, it should be apparent that in the game playing domain, the strategy testing routines will simply be a typical game playing program. The system is made adaptive only through the adaptive routine and the objective function. The only restriction placed on the strategy testing routines is that they must be capable of playing with a variable strategy. That is, the programs must parametrize the playing technique in such a way that it can be varied without reprogramming. Other than this the strategy testing routines can be any game playing simulation.

2.5.2 Testing Using Tree or Graph Search

The playing of any interesting game requires a tree search to help decide upon the best next move. A wide variety of other problems can conveniently be represented by a tree or graph search. Therefore we shall discuss the problems encountered in searching the trees defined by artificial intelligence problems. The main problem we encounter is the size of the tree. For checkers the move tree has been estimated to have 10^{40} nodes, which would take at least 10^{21} centuries to search assuming a node could be investigated in 1/3 of a nanosecond (Samuel(1959)). Clearly some pruning of the tree is required. We will find this is true for virtually all artificial intelligence domains.

2.5.2.1 The Heuristic Search

One of the milestones of artificial intelligence research is the Heuristic Ordered Search developed by Hart(1968), Slagle(1970), and Nilsson(1971). The heuristic search can perform the tree pruning in variable manner ranging from a search of the full tree to a search of only a single path. The heuristic search does this pruning without giving up the theoretical capability of finding the best solution. For this reason we prefer the heuristic search over the many ad hoc search techniques that have none of these theoretical properties. Because we intend to extend the theory of the heuristic search we will present it in its present form along with proofs regarding its theoretical properties. Let me first define the notions of cost and admissibility.

We are searching a tree for a special set of nodes called <u>goal</u> nodes. We associate with each arc in the tree a cost. The cost of a goal is the sum of the arc costs along the path from the root to the goal. The <u>optimal goal</u> is the goal with the least associated cost.

<u>Admissible</u> - A search algorithm is said to be an admissible algorithm if it <u>always</u> finds the optimal goal of any tree. If an admissible algorithm finds a goal g* then cost(g*) ≤ cost(g), for all other goals g in the tree.

□

From this definition we can see that an admissible algorithm is a very general technique since it can find the optimal goal of <u>any</u> tree. Let me now present the heuristic ordered search in detail.

<u>HEURISTIC ORDERED SEARCH</u> Hart(1968), Slagle(1970), Pohl(1970), Nilsson(1971)

Define the following functions for a tree.

```
              □ Root
              |
      ┌-------+-------┐     } g(n) = cost from Root to n
              •
              •
              •
              □ n
              |
          ┌---+---┐         } h(n) = minimum cost from node n
              •                      to a Goal
              •
              •
              □ Goal
```

FIGURE 2.5.2.1a

<u>g(n)</u> - the sum of the arc costs from the root to the node n.

<u>h(n)</u> - the <u>minimum</u> cost path from the node n to a goal.

<u>f(n)</u> = g(n) + h(n) the minimum cost of reaching a goal from the root via node n. Note that the total cost function f is the same for all nodes on the path to the optimal goal.

Given any finite tree g(), h(), and f() are all well-defined, computable functions.

The heuristic search will use the following estimates.

<u>g'(n)</u> - an estimate of g(n). For a tree we will know the true cost g(n) when we reach node n. Therefore in a tree search g'(n) = g(n).

<u>h'(n)</u> - an estimate of h(n). We are forced to estimate h(n) since the true cost from n to a goal depends on the desoendants of n, and we wish to know this cost without having to consider the descendants.

<u>f'(n)</u> = g'(n) + h'(n)

= g(n) + h'(n) since g'(n)=g(n) for a tree.

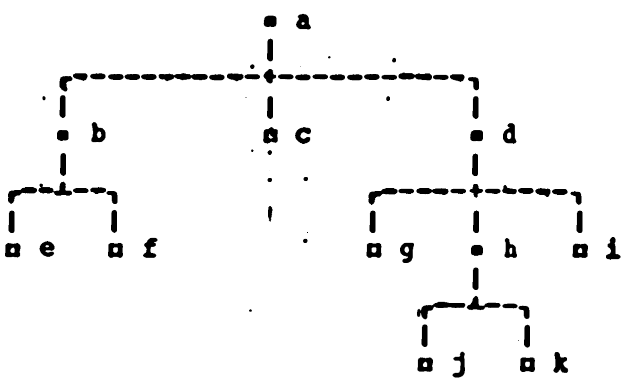

FIGURE 2.5.2.1b

The heuristic search looks for a goal by expanding a <u>search tree</u>, which is originally only the root, until a goal is found. The terminal nodes of this search tree are called <u>tip nodes</u>. The heuristic search keeps track of all paths down the tree by putting the tip nodes of the search tree on a list called the <u>open</u> list. Thus, for the above search tree the nodes e, f, c, g, j, k, and i would be on the open list. The heuristic search will explore next the path whose node on the open list has the lowest <u>estimated</u> cost $f'(n)$. Formally, the algorithm is:

1) put root on open list
2) if $|open| = 0$ stop (failure)
3) find node p, the open node with minimum f' value
4) if $h'(p) = 0$ stop (success) p is the optimal goal.
5) expand p, replace p by its sons on the open list
6) calculate h', g, f', for the sons of p.
7) go to 2)

The heuristic search is admissible if the estimator h' does not overestimate the cost of any nodes. That is, $h'(n) \leq h(n)$ for all nodes n. The algorithm is admissible under this condition because the algorithm expands nodes in order of f'. For nodes in which h has been underestimated, expansion increases the estimate since the g function for the one new arc is exact. Therefore the algorithm will expand all nodes whose estimated cost is less than the true cost of the optimal goal. At that point the optimal goal will be expanded and the algorithm terminates. Let me now formally prove the admissibility of the heuristic ordered search.

Lemma 2.5.2.1
The open list for the heuristic search always contains a node from each path of the search tree that can lead to a goal.

Proof: By induction on the number of nodes expanded by the heuristic search. Call the number of nodes expanded k.

k=0: When 0 nodes have been expanded the root is on the open list, and since the root is on all paths in the search tree, the Lemma is satisfied initially.

k=>k+1: If the Lemma is true after k nodes have been expanded it is also true after k+1 nodes have been expanded. Assume node p is the k+1st node expanded.

Case 1: Node p is a terminal node, but not a goal. Then no path through p can lead to a goal, and the removal of p from the open list does not violate the Lemma.

Case 2: Node p is a goal. Then the algorithm terminates and the Lemma has remained true until termination.

Case 3: Node p is a non-terminal node. Then all the paths in the tree going through node p are covered by the sons of p. Thus, the replacement of p on the open list by its sons maintains the Lemma.

□

<u>Theorem 2.5.2.1</u>

The heuristic search is admissible if $h'(n) \leq h(n)$ for all nodes n. Thus, for the node p which causes termination of the algorithm in statement 4) $\underline{f(p) \leq f(s)}$ for all goals s.

<u>Proof:</u> Assume the algorithm is not admissible and that there exists a goal p* such that $f(p^*) < f(p)$. Consider the selection of node p from the open list in statement 3) just before termination. Since p had the minimum estimated cost of all nodes on the open list at that time, $f'(p) \leq f'(n)$ for all nodes n on the open list. From Lemma 2.5.2.1 we know that a node on the path to p* <u>must</u> be on the open list when p was selected. We call this node n*.

$f'(p) \leq f'(n^*)$	since n* on open when p selected
$g(p)+h'(p) \leq g(n^*)+h'(n^*)$	definition of f'
$g(p)+h(p) \leq g(n^*)+h'(n^*)$	$h'(p)=h(p)=0$, since p is a goal
$g(p)+h(p) \leq g(n^*)+h(n^*)$	bound on h'
$f(p) \leq f(n^*)$	definition of f
$f(p) \leq f(p^*)$	since n* on the path to p*

This contradicts the assumption that $f(p^*) < f(p)$. Thus, there does not exist a goal of lower cost than p.

□

2.5.2.2 Bandwidth Heuristic Search

The heuristic search described thus far is a theoretically well founded algorithm since we have proved it to be admissible. But for many applications practical considerations tempt us to put aside the theoretical value of the heuristic search.

As an example let us consider the Traveling Salesman Problem (Lin(1972), Little(1963)). The problem is to find a minimal cost tour of a group of cities given the inter-city distances. A tour will consist of one visit to each city. This problem, although solvable by the heuristic search, is extremely expensive computationally for large groups of cities, even with very good estimates of h. There is simply a minimum amount of computation that must be done in order to insure an optimal result.

For this reason the recent programs (Lin(1972)) to deal with large scale traveling salesman problems have been designed to find "good" tours in a reasonable amount of computation time. Thus, for some problems we must be willing to consider the tradeoff between the optimality of the result and the cost of calculating it. That is, for this problem we would be satisfied with finding a tour that is say 5% more costly than the optimal tour, if we could save substantially more on the cost of the computation. We would like to modify the theoretical development of the heuristic search to allow for this type of tradeoff.

We must be careful that in loosening the admissibility of the algorithm that we maintain a bound on the additional cost of the resulting solution. Therefore we will define an e-optimal goal to be a goal whose cost is at most e more than the optimal goal (denoted goal*). Thus goal' is e-optimal if

$$f(\text{goal}') \leq f(\text{goal}^*) + e$$

We can find a e-optimal goal by using an estimator $h'(n) \leq h(n)+e$ for all nodes n and $h'(\text{goal})=0$ for all goals. Such an algorithm is said to be e-admissible, since it guarantees finding an e-optimal goal if one exists. Note that an e-admissible algorithm is admissible when e=0.

Theorem 2.5.2.2a

The heuristic search is e-admissible if $h'(n) \le h(n) + e$ for all nodes n, and $h'(goal)=0$ for all goals.

Proof: Call the node the heuristic search terminates with p, and the optimal goal of the entire tree p*. Consider the selection of node p from the open list in statement 3) just before termination. From Lemma 2.5.2.1 we know there exists on the open list a node n* on the path to the optimal goal p*. Thus:

$f'(p) \le f'(n^*)$	since n* on open when p selected
$g(p)+h'(p) \le g(n^*)+h'(n^*)$	definition of f'
$g(p)+h(p) \le g(n^*)+h'(n^*)$	$h'(p)=h(p)=0$, since p is a goal
$g(p)+h(p) \le g(n^*)+h'(n^*)+e$	bound on h'
$f(p) \le f(n^*)+e$	definition of f
$f(p) \le f(p^*)+e$	since n* on the path to p*

Thus, the cost of goal p is at most e above the cost of an optimal goal that was not found by the algorithm because the the heuristic h' could overestimate the cost of a node.

□

But does using an estimate that can be over the true value save any computational effort? Intuitively, if we ever expand a node in which $h'(n) \ge h(n)$ then the algorithm will terminate quickly by going straight down this path because the cost estimate f keeps getting lower and lower and the estimate for this path was already the minimum of the open nodes when it was expanded. One might envision a case in which overestimating the cost function would not save any computation time. For example, if $h1' \le h$ and $h2' = h1'+e$ then one might argue that nothing would be gained, since the order of nodes expanded by the searches using h1' and h2' would be identical. But we have forced all estimates of goals to be zero, so that at some level we must be adding more to our estimate of bad nodes than we are adding to our estimate of good nodes. Thus, a heuristic such as h2' does not fit the required definition.

Empirical tests on the Traveling Salesman Problem indicate that the e-admissible algorithm does indeed decrease the computation time, while adding only slight additional costs to the resulting tour. Using the inter-city distances from Little(1963) (the publication that introduced the Branch & Bound technique on the Traveling Salesman Problem), the e-admissible heuristic search found a tour of cost 65 in less than half the time required to find an optimal tour of cost 63.

The value of the e-admissible heuristic search is that we need not be concerned in overestimating the true cost of a node if we are willing to accept a goal of limited extra cost. This allows the use of computationally cheaper heuristics, and may help the search proceed faster. Thus, we have acheived the desired tradeoff between the optimality of the resulting goal and the cost of calculating it.

Let us consider another practical aspect of the heuristic search. The open list for the houristic search tends to get very large quickly. Since each open node may require substantial storage this can become a limiting factor. For this reason it has been suggested that we employ a "staged" search. This would mean that we run the heuristic search until we run out of storage, then drop off the open nodes with high cost estimates and continue the search process. However, in dropping off the "bad" nodes we may be deleting a path that could lead to the optimal solution. Thus, the staged heuristic search has none of the theoretical properties of the regular heuristic search. Let us now consider how to modify our analysis of the heuristic search to include this practical consideration without forfeiting the admissibility condition.

Let us assume that our cost estimate is always within a bandwidth of (e+d) in estimating the true cost:

$$h(n)-d \le h'(n) \le h(n)+e$$

This is always possible when h(n) is defined for all nodes (i.e., when it is possible to reach a goal from every node.) For the moment let us assume that this is the case, and we will soon discuss the more general case.

In the regular heuristic search, given $f'(n1) > f'(n2)$ does not insure that $f(n1) > f(n2)$. In the bandwidth heuristic search if the heuristic estimators differ by more than a bandwidth then we can decide which true cost is better.

$$f'(n1) - f'(n2) > (e+d)$$

$$f'(n1) > f'(n2) + (e+d)$$

$$f(n1) + e > f(n2) - d + (e+d)$$

$$f(n1) > f(n2)$$

Thus a path through n1 is definitely more costly than a path through n2, and there is no need to put n1 on the open list. We can add a statement to the heuristic search to cut off all open nodes n if $f'(n) > f'(p) + (e+d)$ where p is the node about to be expanded. Thus, we are effecting a staged search at every expansion without dropping the admissibility of the algorithm. Of course, the computation of the bandwidth heuristic may be more costly than that of the regular heuristic. The result is the familiar tradeoff between storage and computation time. At least with the bandwidth search this tradeoff can be considered without forfeiting the admissibility of the algorithm.

BANDWIDTH HEURISTIC SEARCH

1) put root on open list
2) if |open| = 0 stop (failure)
3) find node p, the open node with minimum f' value
4) if h'(p) = 0 stop (success) p is the e-optimal goal.
5) drop all nodes n from open if f'(n) > f'(p) + (e+d)
6) expand p, putting sons on open and removing p
7) calculate h', g, f', for the sons of p.
8) go to 2)

Lemma 2.5.2.2

The open list for the Bandwidth Heuristic Search always contains a node on the path to the optimal goal if $h(n)-d \le h'(n) \le h(n)+e$ for all nodes n.

Proof: By induction on the number of nodes expanded by the Bandwidth Heuristic Search. Call this number k.

k=0: When 0 nodes have been expanded the root is on the open list, and since the root is on the path to the optimal goal the Lemma is true initially.

k=>k+1: If a node n* on the path to the optimal goal p', is on the open list after k nodes have been expanded, then the Lemma is true after k+1 nodes have been expanded.

Case 1: n* is not the k+1st node expanded. Assume node q is expanded at the k+1st step and that n* is dropped from the open list in statement 5). Then

$f'(n^*) > f'(q) + (e+d)$	the "drop" condition
$g(n^*)+h'(n^*) > g(q)+h'(q) + (e+d)$	definition of f'
$g(n^*)+h(n^*)+e > g(q) + h(q)-d + (e+d)$	bandwidth bound on h'
$f(n^*)+e > f(q)-d + (e+d)$	definition of f
$f(n^*) > f(q)$	

Thus, n* is not dropped when another node is expanded.

Case 2: n* is the k+1st node expanded. Here the proof is identical to Lemma 2.5.2.1. One of the sons of n* is also on the path to the optimal goal. Since all of the sons of n* are put on the open list the Lemma remains true after the k+1st expansion.

342

□

<u>Theorem 2.5.2.2b</u>

The Bandwidth Heuristic Search is e-admissible if $h(n)-d \le h'(n) \le h(n)+e$ for all nodes n.

<u>Proof:</u> Assume node p is the goal found by the Bandwidth Heuristic Search in statement 4). Consider the selection of node p in statement 3) just before termination. From Lemma 2.5.2.2 we know a node n* on the path to the optimal goal p* is on the open list at the time node p was selected. Therefore the proof is identical to Theorem 2.5.2.2a since the cost estimate of p is bounded by the cost estimate on n*.

□

We must be particularly careful in defining the heuristic when h(n) is undefined (when there exist some goals from which no goal can be reached). It is fine to say h(n) is undefined or infinite, but what about h'(n)? Since h'(n) is computed by some computer program it must retain some finite value, but it can never be within d of the true cost if h(n) is undefined. Therefore we adopt the convention that if h(n) is undefined $h'(n) \ge N + (e+d)$, where N is some arbitrary large value.

This seemingly innocent definition actually brings about a rather rigid restriction on the Bandwidth Heuristic Search, since seeing a node with $h'(n) > N + (e+d)$ does not necessarily mean that no goal can be reached from this node. Nodes with cost estimate greater than N+e are in an "uncertainty zone" since we do not know whether they are on the path to a goal or not. Expanding a node in the "uncertainty zone" could result in an improper cutoff if the expanded node could not lead to a goal. For this reason we will consider all nodes with $h'(n) > N + (e+d)$ to be nodes with no path to a goal. Therefore we must add the restriction that the bandwidth heuristic search will find an e-optimal goal of cost less than N, if such a goal exists. This is somewhat less clean theoretically, but presents no practical problem since we can choose N to be any number we like.

Given this restriction there is no need to put any node of $h'(n) > N + (e+d)$ on the open list, since even if a goal could be reached via this node it would cost more than N.

Note that the Bandwidth Heuristic Search results in the standard heuristic search when e=0 and d=N.

2.5.2.3 Bandwidth Heuristic Search for MIN/MAX Trees

The bandwidth heuristic search has great practical value for searching trees of the type we have been discussing. We can also make use of the bandwidth heuristic in searching game (MIN/MAX) trees. In this environment the bandwidth heuristic offers the capability of making cutoffs from both ends. That is, bad moves for the program (too high an estimated cost), as well as bad moves for the opponent (too low an estimated cost), need not be considered.

In order to motivate the need for a heuristic search procedure on MIN/MAX trees let us first discuss the special problems involved. We are not generally interested in proving that we cannot lose from a given position. To prove anything about a game tree is always possible, but so time consuming as to render it impractical. Normally, what we desire from the search process is a "good" move to make from our current position. Then when the opponent replies with his move the search procedure should again suggest a "good" move. Thus it is clear we must re-define our theoretical approach since we are no longer interested in guaranteeing optimality.

The standard technique for searching a MIN/MAX tree and finding a good first move is the alpha-beta minimax procedure (Samuel(1967), Nilsson(1971)). This procedure is theoretically intractable for the following reasons.

The evaluation function, although similar to h', is not actually an estimate of some "true" evaluation. If the move tree is extended until all open nodes are terminals with value +N, -N, or 0, then the "true" evaluation of <u>all</u> nodes in the tree is also +N, -N, or 0. Therefore we cannot say the evaluation function estimates the true value within error < N, or else we could tell if a move was guaranteed not to lose. This we know to be impractical even for simple games.

The minimax technique makes a permanent decision on the basis of the evaluation function. We are forced to admit the fallacy of the evaluation since e(n1) > e(n2) does not insure that n1 is a better position than n2. If we were assured of this then we wouldn't need to search the move tree in the first place, we could simply evaluate the sons of the current board position and make the best move. In light of the inexactness of the evaluation function, the finality of the comparisons in the minimax process seems unwarranted.

The alpha-beta procedure extends the move tree to a fixed depth. Thus, it would be impossible to prove anything about its ability to find an optimal result beyond this depth.

All of the techniques that are extensions of the minimax technique, such as the M & N minimax (Slagel(1962)), retain these undesirable features. The fixed and dynamic ordering procedures (Slagle(1970), Nilsson(1971)) help make a guess at the best move, but then use this only to speed the regular minimaxing process.

We would like a routine capable of going to an arbitrary depth in the tree to look into good moves in depth. The heuristic evaluation should <u>order</u> the search but no permanent decisions on the basis of comparing two heuristic values should be made. The most important consideration of the search is not optimality with regard to cost, but accessibility of the resulting goal. Accessibility expresses the notion that each player will make the best move for himself. Therefore some board positions are not accessible because the opponent will not move towards them. All of these ideas can be expressed formally and implemented using the Bandwidth Heuristic Search.

We intend to search a MIN/MAX tree for the optimal, accessible goal by estimating the cost of each node as in the regular heuristic search. We will not be as concerned with finding the optimal, accessible goal as we will be concerned with finding the first move towards such a goal. Also we will emphasize the accessibility of the goal over its optimality. That is, we will look for a goal we __know__ we can get to, instead of one that may be somewhat fewer moves away, but depends on the opponent making an error. Let me now formally define the notions of cost, optimality, and accessibility.

<u>cost</u> $h(n) = 1 + \min(h(m(i)))$ for MIN nodes

$h(n) = 1 + \max(h(m(i)))$ for MAX nodes

where $m(i)$ are the sons of n.

<u>estimated cost</u>

$h'(n) = 1 + \min(h'(m(i)))$ for MIN nodes

$h'(n) = 1 + \max(h'(m(i)))$ for MAX nodes

where $m(i)$ are the sons of non-terminal node n.

$h'(\text{win}) = 0$ and $h'(\text{loss}) > N + (e+d)$ for arbitrary N

for all other tip nodes n:

$$h(n)-d \le h'(n) \le h(n)+e$$

<u>accessible</u> - A node n is accessible if each opponent's move on the path to n is the <u>best</u> move for the opponent. More formally, a node n is accessible if for every MIN node m, on the path to n, $h(m) \ge h(b)$ for all nodes b the brothers of m. NOTE: Accessibility is defined in terms of <u>TRUE</u> cost.

<u>optimal, accessible goal</u> - An accessible goal (denoted goal*) such that $f(\text{goal*}) \le f(p)$ for all accessible goals p.

<u>(e+d)-optimal, accessible goal</u> - An accessible goal p such that $f(p) \le f(\text{goal*}) + (e+d)$.

□

Again we encounter the problem of estimating the cost function when it is undefined. For all losing positions and all nodes that must lead to losing positions the cost function is undefined. We handle the problem in the same way as before: $h'(n) > N + (e+d)$ for a losing node.

We must be careful that our heuristic estimate does not implicitly require knowing whether a given node is a winning or a losing node because we know that this is impractical for any reasonably complex game. This is not the case with the bandwidth heuristic since a node n with estimated cost $h'(n) > N+e$ could lead to either a loss or a win. Once again this will mean we will not find a goal of cost $> N$.

Our strategy will be to expand the minimum cost accessible node two times (on both the MIN and the MAX levels). Then we will evaluate the grandsons and use the bandwidth heuristic to cutoff inaccessible nodes. Then we must recalculate total costs up the tree using the new h' value, at the same time looking for additional cutoffs. Let me stress that due to the bandwidth heuristic we are only dropping nodes that are truly inaccessible, thus maintaining good theoretic properties. Also we are "staging" the process by dropping nodes a bandwidth above the expanded node. The algorithm terminates when we expand a win position, or when all the accessible nodes on the open list have the same first move. In either case we will show that the first move is towards an $(e+d)$-optimal, accessible goal (of cost $< N$, if such a goal exists).

BANDWIDTH HEURISTIC SEARCH FOR MIN/MAX TREES

1) Put root on Accessible Open List (denoted A.O.L.)

2) If |A.O.L.|=0 stop (failure)

3) Select node p from A.O.L. with minimum f' value

4) If h'(p)=0 stop (success) make move towards p

5) Drop nodes n from A.O.L. if f'(n) > f'(p) + (e+d)

6) If 1st moves to 'all' nodes on A.O.L. are the same stop (success) make this move

7) Expand node p on both MIN and MAX levels. Remove p from A.O.L.

8) calculate h' values for grandsons while calculating alpha, the minimax backup value for the two generations of p. Cut off node n and all n's brothers if f'(n) > alpha + (e+d) or if f'(n) > N + (e+d).

9) Put all remaining nodes m on A.O.L. if f'(m) = alpha

10) if alpha = f'(p) then go to 2)

11) recalculate f' values for p and its ancestors

12) cut off nodes n on a MAX level if f'(n) > f'(min) + (e+d), where min is the brother of n with the least f' value.

13) cut off nodes n on a MIN level if f'(n) < f'(max) + (e+d), where max is the brother of n with the greatest f' value.

14) if alpha < f'(p) then recalculate accessibility up the path from p to the root

15) go to 2)

NOTE: A tree structure for the search tree is maintained. Dropping or cutting off nodes permanently removes them from both A.O.L. and the search tree. A.O.L. consists of accessible tip nodes of the search tree. A.O.L. is maintained is steps 9) and 14). These steps require calculations up <u>only 1 path</u> of the tree.

Lemma 2.5.2.3a

The path to the optimal, accessible goal is not dropped from the search tree of the Bandwidth Heuristic Search for MIN/MAX trees if such a goal exists.

Proof: Nodes are dropped from the search tree by being evaluated more than a bandwidth away from a brother on a MIN or a MAX level. The cutoffs used in statement 8) are the same as this, but the calculations are carried out in such a way as to avoid calculating h' for some nodes.

A node n* on the path to the optimal, accessible goal p* cannot be dropped from the search tree on:

Case 1: a MAX level. Assume n* is dropped

$f'(n^*) > f'(min) + (e+d)$	the "drop" criterion
$g(n^*)+h'(n^*) > g(min)+h'(min)+(e+d)$	definition of f'
$g(n^*)+h(n^*)+e > g(min)+h'(min)-d+(e+d)$	bounds on h'
$f(n^*) > f(min)$	definition of f
$f(p^*) > f(min)$	n* on path to p*

This contradicts the optimality of p*. Thus, n* is not dropped on a MAX level.

Case 2: a MIN node. Assume n* is dropped.

$f'(n^*) < f'(max) + (e+d)$	the "drop" criterion
$g(n^*)+h'(n^*) < g(max)+h'(max)+(e+d)$	definition of f'
$g(n^*)+h(n^*)+e < g(max)+h'(max)-d+(e+d)$	bounds on h'
$f(n^*) < f(max)$	definition of f
$f(p^*) < f(max)$	n* on path to p*

This contradicts the accessibility of p*. Thus, n* is not dropped on a MIN level.

Since the root is on the path to the optimal, accessible goal if one exists and since a node on the path to such a goal cannot be dropped on either a MAX or a MIN level, such a node must remain in the search tree.

□

Lemma 2.5.2.3b

The A.O.L. for the Bandwidth Heuristic Search for MIN/MAX tree always contains a node m such that

$$f'(m) \le f(p^*) + e \qquad [1]$$

where p* is the optimal, accessible goal, if such a goal exists and if $h(n)-d \le h'(n) \le h(n)+e$ for all nodes n.

Proof: From the previous Lemma we know that the path to p* is always in the search tree. We show the existence of a node m that satisfies [1] by examining the following two disjoint cases.

Case 1: n* **is** on A.O.L.. Then

$h'(n^*) \le h(n^*)+e$	bound on h'
$g(n^*)+h'(n^*) \le g(n^*)+h(n^*)+e$	add g(n*) to both sides
$f'(n^*) \le f(n^*)+e$	definition of f and f'
$f'(n^*) \le f(p^*)+e$	since n* on path to p*

Thus n* satisfies [1].

Case 2: n* is **not** on A.O.L. There are two conditions that could cause n* to be off A.O.L. The following tree demonstrates these two cases.

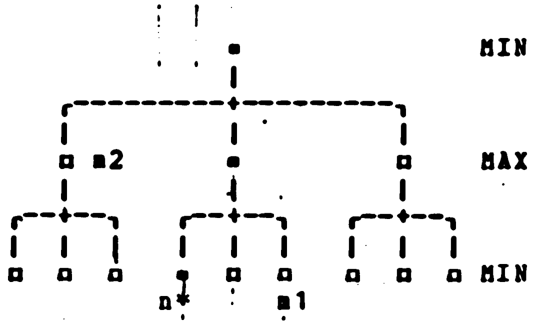

FIGURE 2.5.2.3

We can see that n* can be kept of A.O.L. by a brother on a MIN level (node m1 in the diagram) being <u>overestimated</u>, or by a brother on a MAX level (node m2 in the diagram) being <u>underestimated.</u>

1) a brother on a MIN level is overestimated keeping n* off A.O.L. Then

$f(m1) \le f(n^*)$	since m1 keeps n* off A.O.L.
$g(m1)+h(m1) \le f(n^*)$	definition of f'
$g(m1)+h'(m1)-e \le f(n^*)$	bound on h
$f'(m1)-e \le f(n^*)$	definition of f'
$f'(m1)-e \le f(p^*)$	since n* on path to p*
$f(m1) \le f(p^*)+e$	[1]

Thus m1 is on A.O.L. and it satisfies [1].

2) a brother on a MAX level is underestimated keeping n* off A.O.L.

$f'(m2) \le f'(n^*)$	since m2 keeps n* off A.O.L.
$f'(m2) \le g(n^*)+h'(n^*)$	definition of f'
$f'(m2) \le g(n^*)+h(n^*)+e$	bound on h'
$f'(m2) \le f(n^*)+e$	definition of f
$f'(m2) \le f(p^*)+e$	since n* on path to p*

Thus m2 is on A.O.L. and it satisfies [1].

□

Theorem 2.5.2.3a

The Bandwidth Heuristic Search finds the *first move* towards an (e+d)-optimal, accessible goal of cost < N if such a goal exists where

$$h(n)-d \le h'(n) \le h(n)+e \quad \text{for all nodes } n \text{ and}$$

$$h'(\text{win}) = 0 \text{ and } h'(\text{loss}) > N + (e+d) \text{ for arbitrary } N.$$

Proof: Consider the three statements in which the algorithm can halt.

Statement 2): |A.O.L.|=0 If an optimal, accessible goal of cost < N exists, then by the two previous lemmas, A.O.L. cannot become empty. Therefore, under the conditions of the Theorem the algorithm cannot stop in statement 2).

Statement 6): All nodes on A.O.L. begin with the same 1st move. By the previous lemma, A.O.L. contains a node such that

$f'(m) \le f(p^*)+e$	Lemma 2.5.2.3b
$g(m)+h'(m) \le f(p^*)+e$	definition of f'
$g(m)+h(m)-d \le f(p^*)+e$	bound on h'
$f(m) \le f(p^*) + (e+d)$	definition of f

Thus, the move to m is towards an (e+d)-optimal, accessible goal.

<u>Statement</u> <u>4):</u> Any goal p selected by the algorithm has a cost estimate lower then the node m of Lemma 2.5.2.3b.

$f'(p) \le f'(m)$	since m on A.O.L. when p was selected
$f'(p) \le f(p^*)+e$	Lemma 2.5.2.3b
$g(p)+h'(p) \le f(p^*)+e$	definition of f'
$g(p)+h(p) \le f(p^*)+e$	$h'(p)=h(p)=0$, p is a goal
$f(p) \le f(p^*)+e$	definition of f

Thus p is an e-optimal goal, but it may <u>not</u> be accessible. Assume that along the path to p a brother node b on a MIN level has been underestimated making p appear to be accessible. Then

$f'(b) \le f'(p)$	since p was on A.O.L.
$g(b)+h'(b) \le g(p)+h'(p)$	definition of f'
$g(b)+h'(b) \le g(p)+h(p)$	$h'(p)=h(p)=0$, p is a goal
$g(b)+h(b)-d \le g(p)+h(p)$	bound on h'
$f(b) \le f(p)+d$	definition of f
$f(b) \le f(p^*) + (e+d)$	since $f(p) \le f(p^*)+e$ (above)

Thus, the path through b leads to an (e+d)-optimal, accessible goal. Since the paths to b and p intersect at a MAX level the first move to p is also the first move to b! Therefore the move towards p is a move towards an (e+d)-optimal, accessible goal.

□

Note that this algorithm does not actually find the (e+d)-optimal, accessible goal; it merely assures us that a given move is towards one.

There are many interesting practical aspects of this algorithm. The most important is that it uses the heuristic evaluation to order the search, not to determine the result. If the evaluation mis-orders two nodes then the bad one may be expanded first, but the damage is not permanent as it is in the minimax technique. The bandwidth search only terminates when it knows the heuristic is exact, i.e., a win. It may terminate before this if one move is proved to be better than all others, or if the current position is a losing position.

Secondly the bandwidth search looks only for a good first move. Trying to insure a win by proving a given move will lead to only wins requires too much computation. As an example of just looking for the best move, consider the case of a forced move. The bandwidth search will detect a forced move in the expansion of only one node! Since this is true at any level of the tree, the branching of the tree is restrained.

An important aspect of the bandwidth search as compared to the alpha-beta minimax technique is in the definition of accessibility. For the bandwidth search we define the best move for the opponent in terms of <u>true</u> cost. For the minimax technique the best move for the opponent is determined using the same evaluation function the program uses. Thus, the minimax process implicitly assumes that the opponent will play the same strategy as the program! Clearly this is an invalid assumption, one that is not implicit within the Bandwidth Heuristic Search for MIN/MAX Trees!

The bandwidth search looks deeply into a few moves instead of all moves to a given depth. This is similar to human game playing techniques. It also improves play by spending the time saved by not looking into bad moves on a deeper investigation of more promising moves.

Another important practical feature is that we can vary the bandwidth as the game situation requires. A smaller bandwidth gives a thinner, deeper search as well as a smaller bound on the cost of the final goal. Of course, the cost of evaluating the heuristic will increase, but as the game goes on it is often more possible to analyze board configurations more accurately. Thus, we stand a better chance of finding a goal in the search.

From the proof we see that the goal we are moving towards is (e+d)-optimal. It is under the worst possible conditions that this result is calculated. First, it is under the assumption that a brother of the expanded goal on a MAX level has been underestimated, making our goal inaccessible. But this node could only be underestimated by d. Also we are assuming that the optimal path and our solution path split at the root. If they split anywhere else, then our first move would also be towards the optimal, accessible goal.

Note that this algorithm does not guarantee a win, even if a goal was expanded, unless f(goal) < N+e. This is true since the adding of d to the cost by an underestimated MAX brother could send the cost above $N + (e+d)$, a loss. But this can only be the case when there is no optimal path of cost less than N.

$$f'(p) \geq N+e$$

$$f(m)+e \geq N+e$$

$$f(n^*)+e \geq N+e$$

$$f(n^*) \geq N+e$$

$$f(p^*) \geq N$$

Thus, if the cost of the expanded goal is $< N+e$ then we have proved that we can win from this position. If the cost is in the "uncertainty zone" ($N+e$ to $N + (e+d)$) then we know that an accessible win does not exist within the next N moves.

2.5.3 Evaluation of the Heuristic

We have seen the importance of the heuristic h' in the heuristic search. It controls the order of the search, or in the case of the bandwidth heuristic it controls the final cost of the result as well. Therefore it is important to consider how to evaluate the heuristic. Once again we are into a very problem-dependent domain and cannot say very much about how this evaluation can be done in general. We will instead be concerned with how the known ways of evaluating the heuristic in game playing relate to the rest of the adaptive model.

We stated earlier that a strategy for game playing could be a vector of weights to bias certain important board parameters. We know that the heuristic should predict how many moves are needed to win from a given board position. This is done by evaluating each of the board parameters (such as piece advantage, control of center, guard of king's row) for this position; weighing them by the current strategy and combining them. The obvious way of doing this is the linear polynomial evaluation scheme.

2.5.3.1 Linear Polynomial Evaluation

If (p(1) ... p(k)) are the board parameter detector values and (c(1) ... c(k)) are the weights to indicate the importance of the parameters then the summation of the products c(i)*p(i) constitutes the linear polynomial evaluation function.

We can evaluate a bandwidth heruistic in the same manner except we must shift the result N units to the right to fit the specification for the bandwidth heuristic.

```
|-----------------|------------------|
-N                0                 +N


                       ?-Zone   Loss
|-----------------|---|------|--------|
0                 N  N+e  N+(e+d)    2N
```

FIGURE 2.5.3.1

Thus, the linear polynomial evaluation scheme gives a very intuitive meaning for the heuristic. If the board parameters look so bad (past N+(e+d)) then we assume we cannot recover from this deficit and must lose. The "uncertainty zone" past N+e indicates some doubt that we can recover, but will play the game further in order to find out.

The dimensionality of the strategy space is N since there are N weights to be adjusted. The optimal strategy is a set of weights that correctly evaluates board positions. Therefore every move made by the optimal strategy is the "best" move.

2.5.3.2 Signature Tables

Unfortunately a linear strategy may not be capable of correctly evaluating board positions, because of the interaction of some parameters. For example, the importance of the parameter guard of the king's row in checkers certainly varies with the number of kings the opponent has. Any constant weighting of the parameter cannot take this into account. For this reason a linear scheme may not be capable of accurately rating board positions.

The obvious next step to take would be to introduce second order relations between the parameters. But this would need a strategy space of order N^2 for N parameters. Clearly this would degrade the performance of the adaptive routine by increasing the dimensionality of the search space.

Therefore it has been suggested by Samuel(1967) to selectively relate various board parameters. The researcher should have some feel as to whether two parameters interact or not. Thus, we can handle some of the second order effects without increasing the dimensionality of the strategy space too much. Samuel's technique called "signature tables" uses various levels of tables to account for deeper interaction.

see Figure 2.2a for a diagram of signature tables. Note that when using the signature table evaluation scheme a "strategy" is the set of entries in the signature tables.

2.5.3.3 Use of Memory

In both the linear polynomial and the signature table evaluation scheme we can take advantage of mass memory. Certainly this should be done for frequently encountered board positions, such as those at the beginning of the game. The board positions are stored along with their corresponding parameter values. These parameter values can then be used to evaluate the heuristic using the current strategy. This trades the cost of calculating parameter values for the cost of searching and maintaining the board positions.

Another feature of storing board positions and their associated parameter values is that the accuracy of these parameter values can be made to increase with time. If the values are calculated using a minimax backup for a tree of a given depth, then the next time this board is referenced in a tree search the stored valued can immediately be used. The time saved in this way can be used in searching deeper into the move tree. Thus, the back-up values will become more accurate. In this manner the accuracy of parameters for all board positions stored can become more and more accurate over a period of time.

2.6 Effect of Casting Past Research into the Model

Samuel's checker program

It is hard to criticize a project as successful as Samuel's checker program. However, our study of the adaptive model should give some insight as to why the checker program is as successful as it is.

If we consider the adaptive routine for "generalization learning" we see that what looks like an ad hoc technique is actually theoretically tractable. If the value of m was taken to be n in the formula:

$$c(n+1) = (1-1/m)*c(n) + (1/m)*Z$$

then the first stimulus would have an undesireably large effect on c. On the other extreme, after a large number of trials, the stimulus would have almost no effect on c if m did not stop at 256. This stimulus trainer technique is particularly attractive since the changes make the algorithm computationally faster.

We notice that the objective function in both the "book learning" and the "generalization learning" use 2nd order feedback. This makes it possible for the adaptive routines to more accurately change the strategy as a result of the testing.

In our analysis of the form of the objective function (Section 2.3.3) we noticed that they tend to have a high variance. One criticism of Samuel's program is that his calculation of DELTA (the objective function) is based on only one move. Since the variance of such a small sample is so high this can result in great fluctuations in the adaptation. Thus, one improvement would be to use a strategy over a larger trial set to more accurately rate the strategy. Samuel realized this in his second version. The "book learning" on signature tables was done only after significant data was obtained by examining a number of book moves.

Another improvement would be the use of the Bandwidth Heuristic Search for MIN/MAX trees instead of the alpha-beta minimax procedure.

It is noteworthy that all forms of Samuel's representation of the problem, and all the various types of learning fit quite naturally into the adaptive model.

Klopf's Pattern Recognition Program

Klopf also varied his objective function in order to change the learning characteristics of his system. However, his adaptive routine made no attempt to "predict" a better function from past information. He simply randomly generated a new function and hoped that this would improve performance. This type of ad hoc approach can quite clearly be improved with the use of the adaptive routines developed for this purpose. In particular, the simulated evolution technique of Holland(1969) would replace the poor functions by new functions "close" to those that are currently weighted high.

Fogel, Owens, and Walsh's evolutionary system

Fogel et al attempted to use an evolutionary scheme that will produce new machines "close" to old machines that have done well. In this sense the adaptive scheme is similar to that of Holland's. However, they fail to realize that they are working with a telephone pole space.

The objective function they deal with is a telephone pole space because a slight change in the state transition table can make an infinite change in the language accepted by the machine. Since they measure performance by the language and not the description of the machine, the objective function reflects this discontinuity.

In our study of telephone pole spaces we concluded that the adaptive routine must be specifically tailored to deal with the telephone pole space. That is, only to generate new strategies that will fall on telephone poles. Otherwise a random search of the strategy space is just as good as any other search technique since we cannot make use of past information. Lindsay(1968) noted that this was indeed the case for Fogel's system. A random search of a suitably restricted set of finite machines would have found the resulting machine just about as fast as Fogel's algorithm.

Hopefully, the use of the adaptive model and the analysis presented with it will prevent us from making mistakes such as these when we apply the model to the Natural Language Acquisition Problem.

3. Natural Language Acquisition

3.1 Motivation for Natural Language Aquisition

As a test of the adaptive model we wish to find a challenging application. We will apply it to the Natural Language Acquisition Problem, which is to provide the adaptive mechanisms necessary for a robot to learn to converse with a teacher using natural language. This problem is chosen because we know intuitively, as well as through harsh experience, that it is a hard problem. Also it provides a departure from the rigid structure of game playing, thus showing the generality of the model.

Perhaps the greatest value of the model is that it provides a mechanism for identifying the critical problems involved in a particular application. In the case of natural language acquisition we can see the theoretical reasons why the problem is hard, just by casting the problem into the framework.

In particular, we see that the strategy space is non-numeric. This is true since a strategy will be a transformational context-free grammar which is represented by a character string. From the analysis given earlier we know that this limits our choice of an adaptive routine.

Upon further analysis we see that the objective function is indeed a telephone pole space. This is true since any small change in the grammar (adding, deleting, or changing a production) can lead to an infinite change in the language defined by that grammar. Since the objective function is a measure over the language and not the grammar it reflects this discontinuity. In other words, we could be extremely close to a good grammar, and the objective function would not detect it.

The technique to cope with this problem is to devise operators to be used by the adaptive routine that will only suggest "good" grammars. We wish to traverse the strategy space on the hyperplane defined by these "telephone poles". As we shall see, these operators will come directly from a study of the theory of context-free grammars. We will prove that these operators cover the space (i.e. we can get from one telephone pole to another using only these operators).

Another difficult problem is that of disambiguation. When we actually try to use a grammar in conversation we will find two kinds of ambiguity: lexical ambiguity, which arises from words having more than one meaning and structural ambiguity, which arises when the grammar for a natural language yields more than one parse for a sentence. As we shall see, the heuristic search algorithm provides a convenient means for handling this problem under quite general conditions. Indeed, we must handle it under general conditions because we will have no a priori knowledge of the words, or the grammar. Therefore we cannot even consider handling the problem by treating each case separately, as many previous language systems have done.

3.2 Natural Language Acquisition by Robot

The approach to the Natural Language Acquisition Problem is to try to simulate the conditions under which a child learns a natural language. The robot will be placed in an environment where he can perform certain actions and the teacher will describe these actions to the robot much like a parent describes the world to a child. It should be noted that while we are consciously trying to mimic the conditions under which a child learns a natural language, we are in no way intimating that children learn to understand language in any manner even remotely similar to the way the robot does. However, we will find certain aspects of the robot's outward language capability that do correlate with human behavior.

We desire to have a robot that can walk around a room, store information about the state of the room, and answer questions about the room. We wish to control this activity by giving commands, declarations, and questions in a natural language.

Since the meaning of a sentence in natural language may depend on its context in a given dialog, the robot must maintain some memory of what has been discussed. This "semantic memory" is maintained by storing the meaning of each sentence, as opposed to storing actual word sequences. In fact, the entire parsing, thinking, and speaking of the robot revolves around meaning. The parsing of a sentence is the selection of a possible meaning that is allowed by the grammar and makes sense in the current dialog. This meaning is expressed in some internal form convenient for the robot. The actual words of the sentence are never referenced again. In particular, they are not used by the robot in constructing his reply.

We build the robot with certain innate capabilities, the capabilities required to perform the desired actions. These capabilities are of two types, physical and mental. The physical capabilities may be a motor for moving about the room, a television camera for recognizing objects, or a color wheel for discerning colors. The mental capabilities may be a pathfinding algorithm for figuring how to move about the room; routines for changing the semantic map, etc. The physical capabilities are built as hardware on the robot and the mental capabilities are represented by programs. In particular each mental capability, no matter how simple, is represented by a program, not a dictionary entry.

The fact that these concepts, the physical and mental capabilities, are distinct objects allows us to assign a part of speech (syntactic category) to each. That is, the motors are distinct from the output of the color wheel and thus have a different part of speech. Of course, the robot does not associate the word "verb" with the motors, but assigns them a symbol distinct from the adjectives. This fact that the robot gives a natural breakdown of the parts of speech has led to the phrase "the parts of speech are the parts of the robot". This is one important advantage of introducing the notion of the robot instead of directly programming the computer.

I belabor this point because the robot will be limited to understanding only sentences that involve these concepts. I will indicate how the mental concepts can be extended (Section 3.6.2) after the robot is built, but the physical capabilities are fixed for life. A simple example will emphasize this point. If we build a robot that moves about on wheels then it will never be able to react to a command to jump. This limitation still allows us to use complex grammatical structures, but we must talk about a reasonably simple world. We can talk about simple things in complex ways.

We break the problem into 3 phases. The first phase is to correlate the words of the language to these concepts. The second phase is to infer a transformational context-free grammar to be used by Phase 3 which performs the communication with the teacher in natural language.

One aspect of this approach, and of the adaptive model in general, is that it never says, "I'm done." The robot improves its communication capability with time but never claims to reach the end. The robot is capable of operating in Phase 3 at any time, but its performance reflects the work done in Phases 1 and 2. At any time the robot can re-enter Phase 1 to add new meanings to words, or re-enter Phase 2 to add new structures to the grammar. Thus, improvement comes by the teacher discovering deficiencies during Phase 3 and returning to Phases 1 or 2 to extend the robot's view of the language.

3.3 Phase 1 - Learning the Lexicon

In Phase 1 we want to find a reasonable mapping from the words of the language, the lexicon, to the built-in concepts of the robot. Using this mapping the robot can change a given sentence in the language to concepts. Or, for speaking, the the robot can use the inverse mapping to go from concepts to words. Since the robot associates a part of speech with each concept, the mapping allows the robot to associate parts of speech with the words of the language.

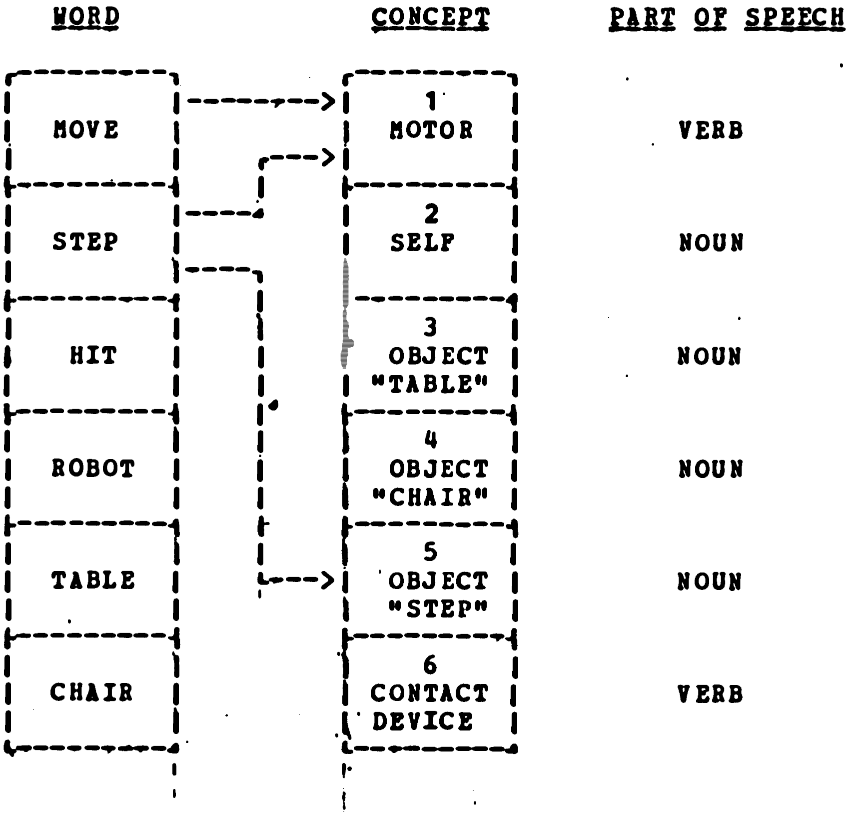

FIGURE 3.3

We make no restrictions on the mapping. A word can map to more than one concept; such as "step" correlating to the concept for moving and also correlating to the concept for a step in a flight of stairs. Also, a concept can map inversely to more than one word. We might allow "move" to also map to the concept for stepping.

We wish to calculate the map under conditions similar to that of a child learning the words of objects in his environment. The robot performs an action and the teacher responds with a description of this action using the language. In this section we discuss the techniques for finding a mapping under these very general conditions.

3.3.1 Conditions for Learning the Lexicon

Let us first consider the conditions under which we expect the robot to learn the lexicon of a natural language. First, the robot is defined by a set of concepts. These concepts, some of which may be implemented by hardware and others by programs, define the robot's capabilities. The only words of the natural language that the robot will ever learn are those words that "mean" what the concept hardware or program does. We are not concerned with the multitude of other words of the natural language that do not have relation to the robot's capabilities.

As we mentioned earlier, we have associated with each concept a part of speech. We intend to find a mapping between the words of the language and the concepts. A word with multiple meanings would be represented by mapping the word to more than one concept. Since the parts of speech are associated with the concepts and not the words, this representation does not restrict a word to a single syntactic category. A given word can be both a noun or verb if it maps to two distinct concepts, one a verb, the other a noun. The mapping can also define synonyms. If more than one word maps to a given concept, then these words are taken to be synonyms when used in this context. Again, the mapping is completely general, as in our earlier example.

We would like the robot to learn the lexicon under conditions similar to those of a child learning vocabulary. As an example consider the situation in which a child has found a pen and begins playing with it. The father responds by saying, "pen." In a like manner, when the robot performs an action, the teacher responds with a description of that action. Since the robot "knows" what it did, it can begin correlating the words in the teacher's description with the concepts composing the action. Since the action is simply a sequence of concepts being executed, it is easy to perform this correlation.

It would be extremely simple to learn the lexicon if all the actions and responses were a single word, as in our example "pen." But we must be able to account for the case where the father says, "red pen." Or the time when the father has some other notion in mind and simply says, "red." Any correlation technique that depends on a one to one relationship between concepts and words in the reply would surely be confused by these latter cases. We are able to cope with this problem since the robot knows all of the concepts involved in the action. The teacher can then reply with any full or partial description of the action.

In Schank(1968) the fact is emphasized that in different natural languages some concepts are overspecified and others are underspecified. This means that certain concepts in French may require more than one word while in English the thought can be expressed in a single word. We encounter a similar problem in working with the robot's concepts. In cases where more than one word is required to express a concept, we connect the words by underscores. Using this convention, they appear as single words to the robot.

The use of the underscore in this fashion helps with the use of idioms as well as simplifying the mapping of multiple words to concepts. Idioms that are groups of words can be denoted as a distinct unit with the use of the underscore. The idiom is then interpreted as one word and mapped to the appropriate concept. Thus we avoid the fruitless effort of trying to combine the individual words into a meaningful concept using the available grammar and parser. Examples of the use of the underscore will appear throughout the sample sentences that follow.

3.3.2 Technique for Correlating Words to Concepts

We maintain a table of cross correlations between words and concepts. Each element of this table contains the correlation coefficient for the word in the row mapping to the concept in the column. For each action these coefficients are biased depending on the words used in the response. If the jth concept is part of the action and the ith word is part of the teacher's reply, then the element c(i,j) is increased. If the jth concept is part of the action and the ith word is <u>not</u> in the response, then c(i,j) is decreased. Similarly c(i,j) is decreased if the jth concept is <u>not</u> part of the action and the ith word is in the response.

The manner in which the coeficients are biased is the "operant reinforcement system" (Skinner(1953), Bush(1955), Minsky(1961)). Using this method we are assured that the limiting coefficients will be the same as the probability of the pairs occuring. This method uses the difference relation:

$$c(n+1) = (1-1/n)*c(n) + (1/n)*Z$$

where n is the iteration number, Z is the amount of bias, taken to be ±1. The problem with this technique is twofold. First, as n gets large the biasing has very little effect. Second, early responses will have extremely large effects on the coefficients. If the early responses are misleading the robot will have trouble overcoming the error, if it ever does.

These problems are exactly the problems encountered by Samuel(1957). His modification of the technique will correct both of these problems. Define m:

$m=16$ for $n<32$

$m=32$ for $32 \leq n<64$

$m=64$ for $64 \leq n<128$

$m=128$ for $128 \leq n<256$

$m=256$ for $256 \leq n$

and use the difference formula:

$$c(n+1) = (1-1/m)*c(n) + (1/m)*Z$$

In this manner the later responses have a larger effect since m stops at 256. Also the early responses have a lessened effect since m starts at 16. The merits of this modification are discused in Minsky(1961):

> "this nicely prevents violent fluctuations in c(n) at the start, approaches the uniform weighting for a while, and finally approaches the exponentially weighted correlation, all in a manner that requires very little computational effort."

At any time during Phase 1 we can calculate the word-concept map from the correlation table. We take as the forward mapping for the ith word those concepts whose coefficients are greater than 1/m. Similarly, the inverse mapping (concepts to words) is calculated by choosing all words in the column whose coefficients are greater than 1/m. If no such element exists in a given row or column we choose the maximum element in that row or column and include this relation in the word-concept map.

Another advantage of this technique is that it provides a probability scale for the meanings of words in common usage. This can be used in speeding the parsing during Phase 3. Many linguistic systems (Winograd(1971)) have used this type of information to speed parsing. Thus, we have a reasonable example of just how such a system could generate this type of probabilistic information by itself.

3.3.3 Sample Execution of Phase 1

This section is a sample execution of the APL/360 program that implements the Phase 1 correlation technique. The sample lexicon and set of concepts are the same as appear in Figure 3.3. The program generates an action performed by the robot and asks for a description of this action. The second line of each action-response pair is typed by the teacher and may be only a partial description of the action, given without regard to syntax. At various times we inspect the correlation table and finally list the word-concept map after several such action-responses. All the indented text is explanatory material and is not part of the program output.

MOVEABOUT

The program that implements the Phase 1 correlation techniqe is executed. Since this sample execution if for a small set of words and concepts, the value of m is initialized to 8 instead of 16.

4 6 ... IS MY ACTION, ENTER DESCRIPTION
HIT CHAIR

This is the first action-response pair. The appropriate correlations are made and the following table results.

	1	2	3	4	5	6
MOVE	0	0	0	-0.125	0	-0.125
STEP	0	0	0	-0.125	0	-0.125
HIT	-0.125	-0.125	-0.125	0.125	-0.125	0.125
ROBOT	0	0	0	-0.125	0	-0.125
TABLE	0	0	0	-0.125	0	-0.125
CHAIR	-0.125	-0.125	-0.125	0.125	-0.125	0.125

The effect of this first action-response is to bias positively the correlation between HIT and CHAIR with concepts 4 & 6. Negative bias is applied to HIT and CHAIR with other concepts as well as to concepts 4 & 6 with other words.

1 5 6 ... IS MY ACTION, ENTER RESPONSE
MOVE HIT STEP

2 5 1 ... IS MY ACTION, ENTER DESCRIPTION
ROBOT MOVE STEP

2 5 ... IS MY ACTION, ENTER DESCRIPTION
ROBOT

Here the teacher responds with only a partial description of the action

5 1 4 ... IS MY ACTION, ENTER DESCRIPTION
STEP CHAIR

3 4 6 ... IS MY ACTION, ENTER DESCRIPTION
HIT TABLE

After these action-responses the correlation table would be as follows.

	1	2	3	4	5	6
MOVE	0.139	-0.111	-0.33	-0.487	-0.004	-0.222
STEP	0.33	-0.195	-0.414	-0.32	0.164	-0.296
HIT	-0.296	-0.487	-0.080	0.004	-0.384	0.33
ROBOT	-0.195	0.234	-0.33	-0.487	0.055	-0.487
TABLE	-0.414	-0.33	0.125	-0.487	0.055	-0.487
CHAIR	-0.222	-0.414	-0.33	0.080	-0.32	-0.246

FORMMAP

The program that calculates the word-concept map from the correlation table. Any element with a correlation coefficient greater than 1/m (.125) will appear in the map. If no such element exists in a given row or column then the maximum element in the row or column is chosen. The result is:

WORD	CORRESPONDING CONCEPT(S)
MOVE	1
STEP	5 1
HIT	6
ROBOT	2
TABLE	3
CHAIR	4

CONC	CORRESPONDING WORD(S)
1	MOVE STEP
2	ROBOT
3	TABLE
4	CHAIR
5	STEP
6	HIT

Notice that the multiples meanings of STEP have been extracted, as well as the fact that both MOVE and STEP can refer to concept 1.

3.3.4 Adding New Words, and New Meanings for Words

It should be clear that by returning to Phase 1, the robot can add new words, or new meanings to words. This is done by simply appending the new words to the correlation table and calculating coefficients for these words from new actions and responses. Again, it is important that the correlation technique use m and not n, since we do not want the new coeficients to become too large at the outset. This is why m starts at 16 and not 1.

This ability to change the robot's view of the meaning of words, as well as an <u>automatic</u> means for introducing new words into the system, makes this approach unique among linguistic systems.

3.4 Phase 2 - Learning the Grammar

In Phase 2 we wish to infer a transformational context-free grammar that will generate a set of sentences given by the teacher. This set of sentences, called the positive sequence or the positive set, is mapped via the map from Phase 1, into the parts of speech. The grammar inferred in Phase 2 should produce this sequence of parts of speech as well as many other sentences that are structurally similar.

Phase 2 directly fits the adaptive model. A "strategy" is the context-free grammar. The objective function is a measure over the languages defined by the grammars reflecting their ability to generate the positive set. The strategy testing section is the parsing of sentences using the grammar.

By casting the problem into the adaptive framework, we immediately become aware of two problems. First, the function space is non-numeric, since the grammars are represented by character strings. Second, the objective function is a telephone pole space because any small change in a grammar (i.e. deleting, adding, or changing a production) can cause an infinite change in the language defined by the grammar. Since the objective function is a measure over the language and not the grammar, it reflects this discontinuity. Both of these problems must be coped with to effectively infer a grammar.

Another important problem considered in this section is that of fitting new sentences into an old grammar. Since we may at some point return to Phase 2 wanting to add new structures to the grammar, we must be able to efficiently isolate the structures of the new sentences that are already contained in the grammar from the structures new to the grammar.

By restricting ourselves to inferring a grammar for a finite set of sentences we avoid the problem of whether a grammar for English exists, and if so whether we could find it. There certainly exist infinitely many context-free grammars to produce a finite set of sentences. Indeed, our problem is to choose a grammar from this infinite set that is not only an efficient grammar, but one that allows us to extract the semantics of the sentence in a convenient way. Herein lies the reason for choosing a transformational context-free grammar over a regular, context-sensitive or systemic grammar. We must be able to associate semantic routines to the productions in order to understand the sentence, and this is more straightforward for context-free grammars than for the other types.

Furthermore, we must be able to associate semantics with the <u>form</u> of the productions, since we have no idea what the grammar is going to be when we begin using it in Phase 3. For this reason I have chosen to work with context-free grammars with certain simple transformations allowed. The benefit we accrue by working under these restrictions, which are forced by an adaptive approach, is that there is no built-in dependency on any special form of sentences, or even any specific language. This is all, of course, under the conditions that the meaning of the sentences can be expressed by the concepts of the robot. This does not mean that there are no limits to this approach, only that the limitations are not clearly visible at the outset.

3.4.1 Handling of this Non-numeric, Telephone Pole Space

Given an initial grammar, it is the purpose of the adaptive routine to modify the grammar trying to improve its performance on a given task. As we discussed in Section 2.4.6, it is critical just how the adaptive routine modifies the grammar. A first guess as to how the adaptive routine could modify a grammar would be to add a production, delete a production, or change a production. This approach would be analgous to the way in which Fogel(1966) modified a finite state machine. As mentioned earlier, operating on grammars in this way introduces a telephone pole space. The language resulting from such a change can be quite different from the input grammar. The resulting grammar may produce none of the positive set, whereas the input grammar produced all of the positive sentences. Therefore, we must define operators that will always maintain similarity between the languages defined by the grammars, as opposed to operators that maintain similarity between the grammars themselves.

We will define operators that can only increase the language defined by a grammar in a restricted way. In this manner, given any grammar that can produce the positive set, the application of these operators will maintain this ability. In the terminology of Section 2.4.6, once on a telephone pole, the application of such an operator would produce another telephone pole. Of course, as we continue increasing the size of the language we may begin producing some negative sentences. The objective function will detect this and rate the grammar correspondingly lower. Let us define operators that will work in this manner for context-free grammars.

3.4.1.1 Definition of Operators

Consider the Chomsky Normal Form of a context-free grammar (Hopcroft(1970)). Any context-free grammar can be represented by productions of the form:

A --> BC where A,B,C are non-terminals

A --> d where d is a terminal

We will use as an example the following context-free grammar in Chomsky Normal Form.

<SENTENCE> --> <NOUN PHRASE> <VERB>
<NOUN PHRASE> --> <ADJECTIVE> <NOUN PHRASE>
<NOUN PHRASE> --> <ADJECTIVE> <NOUN>
<NOUN> --> "BOY"
<VERB> --> "RAN"
<ADJECTIVE> --> "THE"
<ADJECTIVE> --> "BIG"

Consider the parse of the sentence "the big boy ran" using this grammar.

```
                          <SENTENCE>
                              |
                 -------------+-------------
                 |                         |
               <NP>                        |
                 |                         |
         --------+--------                 |
         |               |                 |
         |             <NP>                |
         |               |                 |
         |          -----+-----            |
         |          |         |            |
   <ADJECTIVE>  <ADJECTIVE>  <NOUN>     <VERB>
         |          |         |            |
        THE        BIG       BOY          RAN
```

FIGURE 3.4.1.1

Since the word-concept map provides us with the final row of productions (called instantiations) we need only consider the upper part of the parse tree.

Inferring a grammar is in some sense the reverse of producing a sentence. In our case we are given a sentence and must work up the tree (forming productions as we go), until we have reduced the sentence to a single non-terminal. At each stage of the process we add a production to the inferred grammar that would allow such a reduction to be made. For a Chomsky Normal Form grammar there are only two operations that are required to perform this construction.

Grouping - The grouping operator transforms two adjacent non-terminals (call them B & C) into a new non-terminal (A), and adds to the grammar the production A --> BC.

Viewed in terms of the parse tree, this operator simply moves up the tree replacing one production. The group operator makes no change in the language defined by the input grammar. Its effect is to put a grammar into a more tractable form, as well as to introduce a needed production to derive the positive sentence.

Folding - The folding operator changes the grammar by simply assuming that two non-terminals represent the same structural entity. After making this decision, all occurences of one non-terminal are replaced by the other non-terminal.

This operator is needed for the following reason. The grouping operator always groups two non-terminals into a new non-terminal. If the same structural entity is repeated in the sentence, then some means must be available to relate these different non-terminals that represent this structural entity.

This operator is called folding because we are effectively "folding" the derivation tree over itself at the occurences of these non-terminals. An operation similar to this is used by Feldman(1970) in his grammatical inference technique.

The folding operator has a great effect on the language defined by the grammar. Since the resulting grammar has more productions for the folded non-terminal than did the input grammar, the language is increased. The resulting language clearly contains the input language because no productions have been deleted. Thus, any sentence produced by the input grammar can also be produced by the resulting grammar.

The folding operator is an extremely powerful one. It can quite easily introduce recursive productions into the grammar, making the language it defines infinite. This, in itself, is not bad if the new sentences added to the grammar are acceptable. We can help in this regard by making only selective use of the folding operator. That is to say, we should have good evidence that the two non-terminals being folded really do represent the same structural entity. If we were to apply this operator to randomly selected non-terminals, we may well generate another telephone pole space. For this reason we will develop explicit rules for the use of the folding operator, which are explained in detail in Section 3.4.2. For now we will only be concerned with the theoretical capabilities of the operators.

It should be clear from the picture of the derivation tree in figure 3.4.1.1 that these two operators, grouping and folding, are all that are required to infer a context-free grammar in Chomsky Normal Form. Before we prove this fact in the next section, we introduce another operator added for convenience in the inference process.

<u>Recursion</u> - The recursion operator replaces any repeating non-terminal in the positive sequence by a single <u>new</u> non-terminal. A new production is added to the grammar being inferred to make the original non-terminal recursive.

If another specific non-terminal follows all occurences of the recurred non-terminal then the two are grouped. This is referred to as a "recursive grouping" and is only used to infer the positive closure (*) of the recurred non-terminal.

This operator clearly makes the language defined by the grammar infinite, but only in a very restricted way. Recursion can, of course, be defined by a sequence of groupings, and a fold. However, the use of the recursion operator seems to increase the efficiency of the inference technique. Also, it makes the inferred grammar somewhat simpler. Using the recursive-grouping operator we can include the closure (*) of some non-terminal in one operation. Using only the grouping and folding operators we can only infer the positive closure (+) of a non-terminal. This fact is evident when the resulting grammars for our example sentence are compared. Using only grouping and folding, the definition of a noun phrase is "a sequence of at least one adjective followed by a noun." Using recursion as well as the other operators the definition of a noun phrase is "a (possibly empty) sequence of adjectives followed by a noun." The latter definition includes the case of a noun phrase consisting of only a noun. This additional case must be taken care of, in the former instance, by the addition of another production.

For this reason we include recursion in our list of operators.

3.4.1.2 Proof that Operators Cover the Strategy Space

In this section we will show that the grouping and folding operators are sufficient to infer _any_ Chomsky Normal Form grammar. Since every context-free language can be represented by a Chomsky Normal Form grammar (Hopcroft(1970)) these operators are sufficient to infer a grammar for any context-free language.

Viewed another way, this means that these operators cover the domain of the objective function. Starting from just the positive sentences, _any_ context-free grammar that produces these sentences can be inferred using grouping and folding. (More precisely, only the productions in the grammar that are required to produce the positive sentences can be inferred). Since the "telephone poles" are grammars that can produce the positive sentences this means we can move from an initial telephone pole to _any_ other telephone pole by a sequence of grouping and folding operations. We now prove this result.

Theorem 3.4.1.2 - There exists a sequence of group and fold operations that will infer all of the productions of a Chomsky Normal Form grammar that are used in the formation of the positive sentences.

Proof - We know there exists a sequence of productions in Chomsky Normal Form to generate the positive set, and that this sequence of productions can be represented by the derivation tree. To produce the desired sequence of group and fold operations we alter the derivation tree working from the bottom up and right to left on levels.

1) For every node in the tree using a production of the form A --> BC, we <u>group</u> BC into A (a new non-terminal), adding to the inferred grammar the production A --> BC.

2) For every non-terminal node in the derivation tree that reappears in the derivation of the positive sentence we <u>fold</u> this non-terminal with the non-terminal created by the non-terminals previous appearance.

The resulting sequence of fold and group operators creates a grammar that has all of the productions that are used in deriving the positive sentences. Since the productions are identical (up to a re-naming of the non-terminals) to the Chomsky Normal Form grammar that was used to derive the positive sentences, the languages are identical.

□

Given the derivation tree of the positive set the proof becomes constructive. That is, we have shown that such a sequence of group and fold operators exists by actually constructing the sequence. However, the construction along the lines of the proof requires knowing a grammar for producing the positive set which is exactly what we are trying to find! Thus it is impractical to use this construction as an inference technique. What we have shown is that the group and fold operators are <u>capable</u> of inferring <u>any</u> Chomsky Normal Form grammar. If we were not assured of this fact, it would be unwise to even attempt to use these operators in an inference technique.

Clearly, the order of application of the operators is critical. This choice of when to apply what operator is left to the adaptive routine. If the adaptive routine makes an imprudent choice then the objective function should detect this error and bias the results. Therefore we need not proceed directly to the "correct" grammar, we can infer it by trial and error.

It is important to note that this inference technique is effective because we have disregarded the infinite number of grammars that cannot produce the positive set. The grammars that can produce the positive set are generally referred to as "deductively admissible" grammars (Horning(1969)). But more important than this, we have defined the "heuristic connection" (see Section 2.4.6) of the grammars to be defined by a closeness in the languages defined by the grammars, not by a closeness in the character string representation of the grammars.

This is of critical importance because we measure a grammar's value by the language it defines. If we search the grammar space by moving about in steps that keep the grammars "close" (by adding, deleting, or changing productions) then we are searching a telephone pole space. If we search the grammar space by moving about it in steps that keep the languages "close" then we can employ hill climbing techniques.

3.4.2 Inference Technique

We now express the inference technique in terms of the adaptive model.

strategy - a transformational context-free grammar. (the only transformations allowed will be simple permutations used in Phase 3, see Section 3.5.2.5)

objective function - Due to the operators used, all strategies (grammars) to be tested are deductively admissible, therefore we need not perform any parsing during the test. The objective function is proportional to the grammar's ability to be associated with the semantic routines (this is discussed in Section 3.5.3), and is somewhat sensitive to the size of the grammar.

adaptive routine - The Simulated Evolution Technique (Section 2.4.5) using the grouping, folding, and recursion operators.

The inference technique consists of forming an initial grammar from the positive sentences and then applying the operators to this initial grammar to create new grammars. We will first discuss the formation of the initial grammar and then the decision rules that control the application of the operators.

To show the formation of the initial grammar consider the positive sentence "the big boy ran". The inference technique would use the word-concept map to change all of the words to parts of speech. This would result in

J J N V

Where "J" stands for adjective, "N" for noun, and "V" for verb. This will serve as the TERMINAL alphabet for the inference technique. This sequence of terminals is used to form the initial grammar by creating non-terminals that will produce each terminal symbol. All other non-terminals will result from the application of the operators. The initial grammar is:

. --> <ADJ><ADJ><NOUN><VERB>
<ADJ> --> J
<NOUN> --> N
<VERB> --> V

The initial grammar produces exactly the positive sequence, and thus, according to our initial formulation, corresponds to a telephone pole. From this initial grammar the operators can move us from telephone pole to telephone pole until a satisfactory grammar is found.

Given a specific grammar and an operator to be applied, the adaptive routine must select specific non-terminals to be operated upon. The following rules are used in making this determination.

Grouping - Among all the starting productions (those with a punctuation symbol on the left-hand side) the grouping operator groups those two non-terminals which appear _in sequence_ most frequently.

Folding - Among all the non-starting productions, the non-terminals with production schemes of the type appearing below will be folded.

```
A --> B        A --> BM
  AND     OR     AND
C --> B        C --> BN
```

In the above examples A and C would be folded. If more than one such pair exists, one pair is randomly selected.

Recursion - Among the starting productions, any non-terminal that appears twice or more in a row can be made recursive. If more than one such non-terminal exists, one is randomly selected.

Although we are not particularly striving to find the most efficient grammar for producing the positive sentences, there are certain productions that must be avoided. An example is a production of the form A --> A, which occurs frequently as a result of the folding operation. Also there is no need for two identical productions to be in a grammar. For these reasons we employ a "cleanup" routine after every operator is applied. This cleaning up process weeds out all such needless productions.

The cleanup routine also helps the inference process in two more subtle ways. First, the case of two similar starting productions.

. --> <NP> <VP> <PP> (1)

. --> <NP> <VP> <J> (2)

Productions like these cause the grammar to be extremely inefficient. Consider a sentence that could be parsed using production (2). If the parse technique first tried to parse it using (1) it would be successful until it tried to find a <PP>, at that point it would fail. Then the parser would try to parse it using (2) and would have to re-perform all the work done in parsing <NP> and <VP>. Since the work involved in parsing these non-terminals can be substantial (each may involve many words), productions of this type can cause extreme inefficiencies.

To handle this problem the cleanup routine changes the productions to

. --> <NP> <VP> <MOD>

<MOD> --> <J>

<MOD> --> <PP>

Using these productions the parser can parse <NP> and <VP> and then "select" a <PP> or a <J>. When the cleanup routine makes such a change it prints a message indicating such a "selecting" process has taken place.

In a similar situation it may appear that one of the two non-terminals to be selected is a more general version of the other. In this case the cleanup routine "absorbs" one non-terminal into the other. The effect of this is to generalize the notion of this syntactic entity in other "incidental" productions throughout the grammar. An incidental production is one in which the left hand side is neither of the two non-terminals being absorbed. Consider the following example.

```
         . --> <NP>      <VP> <MOD>      (1)
         . --> <NEW NP> <VP> <MOD>       (2)
  <NEW NP> --> <NP>                      (3)
  <NEW NP> --> <NP> <PP>                 (4)
      <PP> --> P <NP>                    (5)
```

We notice that <NEW NP> and <NP> could be "selected" (from their analgous appearance in the two starting productions) and that <NEW NP> seems to be a more complex version of <NP> (from production (3)). We therefore generalize <NP> and <NEW NP> in all incidental productions. The grammar resulting from "absorbing" <NP> into <NEW NP> would be:

```
        . --> <NEW NP> <VP> <MOD>
   <NEW NP> --> <NP>
   <NEW NP> --> <NP> <PP>
       <PP> --> P <NEW NP>
```

The most significant change, aside from the deletion of production (1), is that the definition of <PP> has been generalized by the use of the more complex noun phrase.

Before the grammar is passed to Phase 3 for use in a dialog one more cleanup step is employed. In this step the instantiating productions are backed up wherever possible to speed parsing.

3.4.3 Sample Inference

In order to demonstrate the use of the operators we will apply them in the order suggested by Theorem 3.4.1.2 to infer the given grammar exactly. Starting with the initial grammar, the inference technique would procede as follows.

```
                 .                           . --> <ADJ><ADJ><NOUN><VERB>
                 |                       <ADJ> --> J
       +------+--+--+------+            <NOUN> --> N
       |      |     |      |            <VERB> --> V
     <ADJ>  <ADJ> <NOUN> <VERB>
```

1) GROUP <ADJ> and <NOUN> into <NP>.

```
                 .
                 |                           . --> <ADJ><NP><VERB>
       +---------+---------+              <NP> --> <ADJ><NOUN>
       |         |         |             <ADJ> --> J
     <ADJ>      <NP>     <VERB>         <NOUN> --> N
                 |                      <VERB> --> V
              +--+--+
            <ADJ> <NOUN>
```

2) GROUP <ADJ> and <NP> into <NP2>.

```
                 .
                 |                           . --> <NP2><VERB>
          +------+---------+             <NP2> --> <ADJ><NP>
          |                |              <NP> --> <ADJ><NOUN>
        <NP2>            <VERB>          <ADJ> --> J
          |                             <NOUN> --> N
     +----+----+                        <VERB> --> V
     |         |
   <ADJ>      <NP>
               |
            +--+--+
          <ADJ> <NOUN>
```

3) <u>FOLD</u> <NP> and <NP2>.

```
             •
             |                              • --> <NP> <VERB>
         ┌───┴────────┐               <NP> --> <ADJ><NP>
         |            |               <NP> --> <ADJ><NOUN>
        <NP>        <VERB>           <ADJ> --> J
         |                          <NOUN> --> N
   ┌─────┴───┐                      <VERB> --> V
   |         |
 <ADJ>      <NP>
             |
          ┌──┴──┐
        <ADJ> <NOUN>
```

FIGURE 3.4.2

3.4.4 Adding New Sentences to an Old Grammar

At some time we may wish to return to Phase 2 to increase the complexity of the grammar. Automatically changing the grammar to add these new syntactic structures brings about certain new problems. We wish to add the new structures contained in the new positive sentences, but not to add the structures already contained in the grammar.

A first attempt at adding new sentences to an existing grammar might be to begin parsing the new sentences with the old grammar. Since the new sentences may not be in the language defined by the old grammar, the parse may fail. At this point one might try to "guess" where to begin parsing again. Attempts along these lines have failed because it is hard to decide where to begin parsing again. For this reason we approach the problem in a different way.

We maintain a list of operators used in transforming the initial grammar to the current one. Then when new positive sequences are to be included in the grammar, we apply these "old operators" to the new sequences. In this manner we extract from the new sentences the structures already contained in the grammar, and they are extracted <u>independent</u> of left-to-right ordering. What remains of the new sequences are the new structural entities. These are added to the grammar by the adaptive process in the manner described in Section 3.4.2. That is, by continued use of the grouping, folding, and recursion operators.

What we are in fact building is a hierarchy of specific operations. Since the robot is presented with increasingly complex sentences, this hierarchy is related to the complexity of the structures. When presented with a new, more complex sentence to be added to the grammar, the robot extracts all the "simple" structures that it is already familiar with by applying the old operators. With the new structures isolated the robot can clearly see what new syntactic structures are contained in the new sentence and add them to the grammar.

3.4.5 Sample Execution of Phase 2

This section is a sample execution of the APL/360 program that implements the Phase 2 grammatical inference technique. The teacher types a set of sentences for which he wishes the robot to infer a grammar. These are the positive sentences, which are then changed to parts of speech using the word-concept map. From these sequences of parts of speech the initial grammar is formed. The adaptive routine then applies operators resulting in new grammars.

The indented text is explanatory material and is not part of the program output. The only words typed by the teacher are the original sentences, all other printing is done by the program.

```
     ADDTOGRAMMAR
            The program to ask for positive sentences.


A BIG TABLE IS AT POSITION 22.
A CHAIR IS RIGHT_OF THE TABLE.
WHAT IS IN POSITION 23?
MOVE TOWARDS IT!
0
```

The positive sentences are entered by the teacher. Note that no analysis of these sentences is performed by the robot. The resulting sequences of parts of speech are then requested and printed.

```
     POSEQ
.JJNVPJN
.JNVPJN
?NVPJN
!VPN
```

```
     STARTUP
            The program to form the initial grammar.


INITIAL GRAMMAR:
.-->1123412
.-->123412
?-->23412
!-->342
2-->N
3-->V
4-->P
1-->J
```

Note that all the <u>numbers</u> <u>are</u> <u>non-terminals</u> and the letters are terminals. This grammar produces exactly the above sequences.

EVOLVE

The adaptive routine program will apply the operators to the initial grammar.

```
GROUPING 12 INTO 2

PRODUCING GRAMMAR:
.-->2342
?-->2342
!-->342
3-->V
2-->N
4-->P
1-->J
2-->12
```

This is a recursive grouping of 1 & 2, the robot now has the notion of a simple noun phrase (non-terminal 2). Note that first two sentences were made structurally identical by this grouping and thus one production was dropped.

```
GROUPING 42 INTO 6

PRODUCING GRAMMAR:
.-->236
?-->236
!-->36
2-->N
3-->V
4-->P
1-->J
2-->12
6-->42
```

The robot now has the notion of a prepositional phrase (non-terminal 6). At this point the starting productions are amenable to the semantic routines (see Section 3.5.2.4) and the inference terminates. The grammar is now "cleaned up".

```
THE FINAL GRAMMAR (AFTER BEING CLEANED UP) IS:
BACKING UP P
BACKING UP J
.-->236
?-->236
!-->36
2-->N
3-->V
2-->J2
6-->P2
```

The list of operators used thus far is maintained to help add to the grammar at a later date.

THE OPERATORS USED TO THIS POINT:
R121
G122
G426

Putting the inferred grammar in a more readable form.

```
   .--><NP><VP><PP>
   ?--><NP><VP><PP>
   !--><VP><PP>
<NP>-->J*N
<PP>-->P<NP>
<VP>-->V
```

We could now use this grammar in a dialog. In fact, we use this grammar at the beginning of the sample dialog in Section 3.5.6. We can also add new grammatical structures to the grammar by adding new positive sentences.

```
     ADDTOGRAMMAR
THE BIG TABLE IS SQUARE.
THERE IS A PIANO NORTH_OF THE CHAIR.
IN POSITION 48 THERE IS A CHAIR.
IT IS SMALL.
WHAT IS SOUTH_OF THE PIANO?
WHERE IS IT?
QUICKLY WALK TO THE TREE!
IS THE TREE BIG?
IS A TABLE AT LOCATION 22?

     POSEQ
.JJNVJ
.AVJNPJN
.PJNAVJN
.NVJ
?NVPJN
?AVN
!AVPJN
?VJNJ
?VJNPJN
```

ADDGRAMMAR

This program applies the old operators to the new sequences and adds the new productions to the old grammar.

```
CHANGING THE FOLLOWING GRAMMAR...
.-->236
?-->236
!-->36
2-->N
3-->V
4-->P
1-->J
2-->12
6-->42
```

```
THE RESULTING GRAMMAR IS:
.-->231
.-->7326
.-->6732
.-->231
?-->236
?-->732
!-->736
?-->321
?-->326
.-->236
!-->36
2-->N
3-->V
7-->A
4-->P
2-->12
6-->42
1-->J
```

Notice the many redundant productions, these will be deleted after the first operator is applied.

```
     EVOLVE
GROUPING 73 INTO 8

PRODUCING GRAMMAR:
SELECTING 9-->1|6
SELECTING 0-->8|3
.-->209
.-->029
.-->902
?-->209
?-->02
!-->09
?-->029
2-->N
3-->V
7-->A
4-->P
2-->12
6-->42
1-->J
8-->73
9-->1
9-->6
0-->8
0-->3
```

The notion of a verb phrase is represented by non-terminal 0. Two "selections" are applied in the cleaning up process. This grammar is then cleaned up before being used in the dialog in Section 3.5.6. The resulting grammar is written in a more readable form.

```
    .-->&lt;NP&gt;&lt;VP&gt;&lt;MOD&gt;
    .-->&lt;VP&gt;&lt;NP&gt;&lt;MOD&gt;
    .-->&lt;MOD&gt;&lt;VP&gt;&lt;NP&gt;
    ?-->&lt;NP&gt;&lt;VP&gt;&lt;MOD&gt;
    ?-->&lt;VP&gt;&lt;NP&gt;
    ?-->&lt;VP&gt;&lt;NP&gt;&lt;MOD&gt;
    !-->&lt;VP&gt;&lt;MOD&gt;
 &lt;NP&gt;-->J*N
 &lt;PP&gt;-->P&lt;NP&gt;
 &lt;VP&gt;-->AV | V
&lt;MOD&gt;-->&lt;PP&gt; | J
```

THE OPERATORS USED TO THIS POINT:
R121
G122
G426
G738

This process of adding new sentences thereby making the structural content of the grammar more complex is continued to include sentences of the type used in Section 3.5.6.

For sessions 3, 4 & 5 in the dialog in Section 3.4.6 the following grammars were inferred by this inference technique. In most cases the positive sentences were exactly the sentences used in the dialog. All the grammars have the same starting productions which follow.

```
. --> &lt;NP&gt;&lt;VP&gt;&lt;MOD&gt;
. --> &lt;VP&gt;&lt;NP&gt;&lt;MOD&gt;
. --> &lt;MOD&gt;&lt;VP&gt;&lt;NP&gt;
? --> &lt;VP&gt;&lt;NP&gt;&lt;MOD&gt;
? --> &lt;VP&gt;&lt;NP&gt;
? --> &lt;NP&gt;&lt;VP&gt;&lt;MOD&gt;
! --> &lt;VP&gt;&lt;PP&gt;
```

SESSION 3

```
 <NP> --> <SNP>
 <NP> --> <SNP><PP>
<SNP> --> J <SNP>
<SNP> --> N
 <VP> --> <SVP>
 <VP> --> V
<SVP> --> A <VB>
 <VB> --> V
 <PP> --> P <NP>
<MOD> --> J
<MOD> --> <PP>
```

SESSION 4

The following productions are added to the above grammar by the inference process.

```
    <NP> --> <SNP><CLAUSE>
<CLAUSE> --> R <CPRED>
 <CPRED> --> V <MOD>
```

SESSION 5

The following productions are added to the grammar and all occurences of <NP> in the starting productions are replaced by <COMPLEX NP>.

```
        <NP> --> N <CPRED>
<COMPLEX NP> --> A <SV>
<COMPLEX NP> --> <NP>
        <SV> --> <NP><VB>
```

3.5 Phase 3 - The Dialog

Phase 3 is the operational phase of the robot. Using the word to concept map developed in Phase 1, and the transformational context-free grammar inferred in Phase 2, we wish to converse with the robot. The conversation will consist of declarations containing information about the state of the environment, questions asking for information about the environment, and commands to perform certain actions and thereby change the state of the environment.

Given a context-free grammar and a sentence, it is easy to decide if the sentence fits the grammar. Our task is more difficult than this, we must see if the sentence fits the grammar, but we must also extract the meaning of the sentence as we parse it. Since the Phase 3 machinery must be able to work with any grammar produced by Phase 2, we must develop some <u>autcmatic</u> means of associating semantic routines to the productions of the grammar.

Once we have associated semantic routines to each production, we can parse the sentence using the standard context-free parsing techniques. This task is again complicated by the fact that we do not know the specific type of grammar we are working with. That is, we cannot assume it is simple precedence or LR(k). We must also decide what semantic information need be stored as we parse.

The parsing task is also complicated by ambiguities which stem from two types: lexical ambiguity which arises from words having more than one meaning and structural ambiguity which arises when a grammar yeilds more than one parse of a sentence. We must be concerned with this problem since we cannot be assured that Phase 2 will infer an unambiguous grammar. Indeed, Phase 2 <u>must</u> infer an ambiguous grammar given certain positive sentences. This, of course, stems from the fact that natural languages <u>are</u> ambiguous. We will discuss ways of handling both forms of ambiguity, but first I should point out that the robot must not only be theoretically capable of coping with ambiguities, but cope with them efficiently. If every word in a 10 word sentence has 3 possible meanings then there are 3^{10} possible lexical interpretations (not to mention structural ambiguity). It is clear that the parsing technique must be designed to avoid this exponential growth.

Another problem is the use of the semantic map, the storage device for the meaning of sentences. The robot must store information about himself and the environment and he must be capable of extracting information from the map. The semantic map must interface with the parser in order to weed out certain ambiguities. We will deal with all these problems in this section.

Another problem we will discuss is that of speaking. By speaking we mean the mechanism by which the robot formulates and communicates its replies, <u>not</u> the actual acoustic synthesis of speech. All communication with the robot is done by means of a typewriter console and thus the "speaking" of the robot consists of its formulation of replies and typing them out. We wish for the robot to improve its speaking capability with time by using the same grammar used for parsing. The first problem is that of getting the robot to generate the concepts he wishes to communicate to the teacher and then order these concepts into a legal sentence using the grammar. It turns out that the ordering of the concepts by the grammar takes 2-3 times the effort of parsing the same sentence. This should not be too disturbing since it agrees with human experience. This method of speaking, although more costly, is more natural than using programmer formatted output, or a reordering of the input words; techniques which are common in other natural language programs. Once again, this method of speaking is due to an adaptive approach to the problem, since we initially have no knowledge of the exact lexicon or grammar.

The final topic of this section is a discussion of ways to extend the linguistic system. We will consider ways of extending the grammar directly from the dialog, and a way to extend the robot's capabilities without reprogramming.

3.5.1 Basic Parse Technique

We are faced with some rather unique problems related to parsing with the grammar inferred in Phase 2. First, we do not know the type of the grammar we are dealing with. We cannot depend on the grammar being simple precedence, SLR(1), LR(k), or any of the other types of grammars for which efficient automatic parsing techniques have been developed (DeRemer(1969), Gries(1971)). Therefore, these well developed techniques cannot be applied to our problem. Second, we are not even assured that the grammar inferred in Phase 2 is unambiguous. In fact, it can be argued that given certain positive sentences, Phase 2 <u>must</u> infer a ambiguous grammar. If this is the case then our parsing technique must be capable of finding all the ambiguous interpretations of a sentence.

For these reasons we have chosen to develop a general parsing technique based on the Bandwidth Heuristic Search (Section 2.5.2.2). We will take advantage of the inherent parallelism of this search process which gives us the capability to carry forward several parses (possible ambiguities) at the same time. The heuristic will control the order in which the parses are carried forward. Hopefully it will cause the search to go directly to the correct parse without needlessly considering other choices. With these ideas in mind let us discuss exactly how we form the Bandwidth Heuristic Search into a general parsing technique for context-free grammars.

3.5.1.1 Parsing using the Bandwidth Heuristic Search

The Bandwidth Heuristic Search is a tree searching algorithm discussed in detail in Section 2.5.2.2. We use the heuristic search to search the tree defined by a context-free grammar for a left-most derivation of the sentence being parsed. It is a top-down parsing technique. Let us formally define these terms.

<u>derivation</u> - a sequence of productions of the grammar that transform the sentence (start) symbol of the grammar into the desired sentence.

<u>sentential form</u> - any string of terminals and non-terminals that can be derived from the sentence symbol.

<u>left-most derivation</u> - a derivation in which only the left-most non-terminal in each sentential form is transformed by a production.

<u>tree defined by a context-free grammar</u> - the tree consisting of:

1) the root is the sentence symbol
2) the nodes are sentential forms
3) the sons of any node are the sentential forms resulting from applying productions to the leftmost non-terminal of the node.

As an example consider the following grammar and the tree it defines.

<.> --> <NP><VP><PP>

<NP> --> J <NP>

<NP> --> N

<VP> --> V

<PP> --> P <NP>

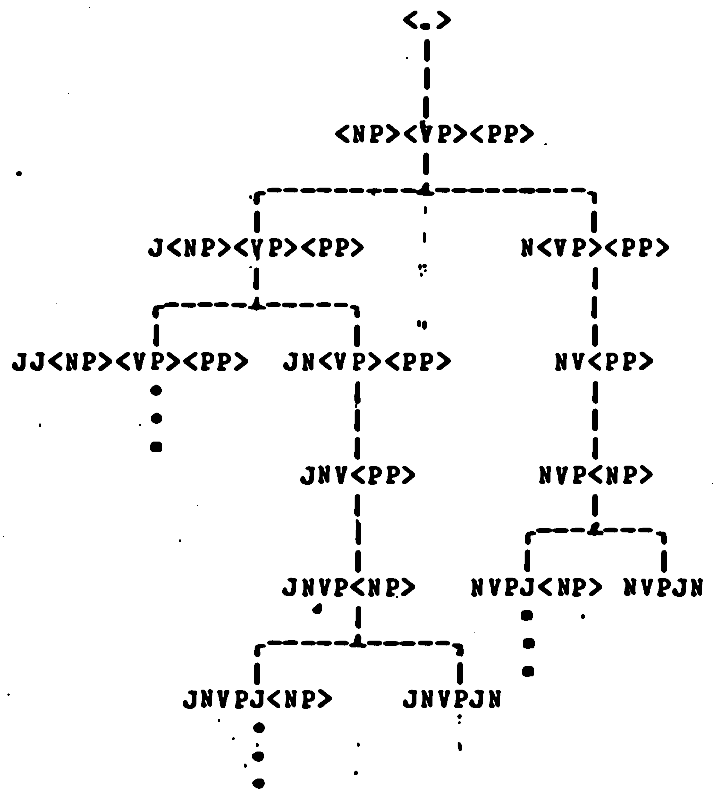

FIGURE 3.5.1.1

From these definitions we can immediately apply the Bandwidth Heuristic Search to the problem. We "expand" a node (a sentential form), thereby creating its sons, by applying <u>all</u> the productions for the leftmost non-terminal of the sentential form. This generates the tree defined by the grammar as the search progresses.

The cost function g(n) is defined as 1 unit per arc in the tree, or one unit per production used in the derivation. The heuristic h'(n) is the estimated cost of deriving the "goal sentence" (the sentence we are parsing) from a sentential form. Clearly this is a function of how many non-terminals are in the sentential form, since each non-terminal requires at least one production to reduce it to a terminal. The heuristic h' can be set to some arbitrarily high value in either of the following cases.

1) If the terminals left of the leftmost non-terminal do not match the corresponding terminals of the goal sentence. This sentential form can never produce the goal sentence, since these terminals cannot change.

2) If the sentential form is longer than the goal sentence. Since the lengths of sentential forms for a context-free grammar are non-decreasing (Hopcroft(1970)), the goal sentence cannot be derived from a longer sentential form.

In either of the above cases, the sentential forms will be dropped from the search because their cost will be outside the bandwidth defined for the heuristic. The heuristic h' is defined as zero when the sentential form matches the goal sentence identically, in which case we have found the goal sentence and the path from the root to the goal is a legal parse of it.

This parse technique is a top-down parse and therefore is similar to other top-down parsing techniques (Gries(1971)). However, one cannot easily define the notion of "backing up" for this technique. When a particular parse fails other parses are tried, but not always one that is closely related to the one that just failed. The purpose of the heuristic is to try to select the most promising parse and continue along that line. In many cases this will result in changing only the last choice but, as we shall see when we consider ambiguities, the most promising parse can quickly change from one interpretation to a totally different interpretation. The heuristic search gives us this flexibility, whereas using an automatic backup technique, such as using an alternate production for the last non-terminal expanded, does not allow for this type of flexibility.

3.5.1.2 A Sample Basic Parse

Let us now consider a sample sentence to be parsed by the robot. At this point we will assume no ambiguity in order to make the basic parsing technique more apparent. In Section 3.5.3 we will discuss the full parse technique which allows for ambiguities. The basic parsing technique would proceed as follows.

1) The teacher types in a sentence such as "A table is left-of the chair."

2) Using the word-concept map, these words can be changed to concepts and thus to parts of speech. Since we have ruled out ambiguities for now, each word can map to only on part of speech. The result is <u>JNVPJN</u>, which is the goal sentence. Note that for the parser the <u>terminal symbols are the parts of speech</u> not the actual words of the sentence.

3) We begin the heuristic search using the start symbol <.> since this is the punctuation given with the sentence.

```
                              <.>
                               .
                               .
                               .
                         <NP><VP><PP>
                               .
                   ..........------.
                   .               |
              J<NP><VP><PP>    N<VP><PP>*
                   .
          .--------..........
          |                 .
   JJ<NP><VP><PP>*     JN<VP><PP>
                            .
                            .
                            .
                          JNV<PP>
                            .
                            .
                            .
                          JNVP<NP>
                            .
                    ........-------.
                    .              |
               JNVPJ<NP>         JNVPN*
                    .
             .------........
             |             .
       JNVPJN<NP>*       JNVPJN
```

* - Denotes nodes cut off by the bandwidth since they cannot produce the goal sentence.

FIGURE 3.5.1.2

In this example we see that the heuristic search would go <u>directly</u> to the correct parse, since there would be only 1 node on the open list at any time, the others being dropped by the bandwidth criterion. We will find that this is generally the case for the parsing of most simple sentences. Note that this is independent of the particular values of h' assigned to a node. The true use of the heuristic will not be evident until we consider more complex grammars. The heuristic will become even more important when we parse ambiguous sentences.

3.5.2 Extracting Meaning from Sentences

At this point it should be clear exactly how the robot syntactically analyzes a sentence by searching the tree defined by the grammar. We have not mentioned what the robot does with this analysis and how the meaning of the sentence is extracted during the parse. In this section we discuss exactly how the robot "understands" a sentence.

The meaning of a sentence is dependent upon two factors, the meaning of the individual words composing the sentence, and the way in which these words are structurally related. We have already discussed how the meaning of the individual words is expressed within the the concept program related to that word. The meaning of the sentence contained within the structural relationship of the words is extracted during the syntactic parse of the sentence.

We perform this semantic analysis during the parse in a manner similar to the semantic analysis for computer languages (Gries(1971)). This is done by associating a semantic routine (a program) to each production in the grammar. When a production is used in the parse of the sentence, the corresponding semantic routine is executed.

The remainder of this section will deal with the internal representation of semantics, and the manner in which the semantic routines, working in conjunction with the concept programs, change the natural language sentence into this representation.

3.5.2.1 The Semantic Map

The semantic map (or semantic store) is a storage device for the meaning of sentences. The form of such a store for general artificial intelligence capability has been debated rather heatedly in recent years (Quillian(1966), Fodor(1969)). Once again we can simplify the problem because we are working within a very restricted range of capabilities.

It is sufficient for our purposes to have the semantic map store the state of the robot's environment. The semantic map will be changed as a result of a declaration telling the robot something about the room. The semantic map will be searched to find answers to questions and to help follow commands. In this way the semantic map serves as a memory device for the entire dialog. Nothing related to exact word sequences is stored, only the meaning of the past sentences is remembered. More precisely, only the meaning of the sentence as it pertains to the environment is stored. For this reason, the robot will not remember the fact that a question was asked because a question does not change the semantic map.

The semantic map may also be changed as a result of the robot's experiences, as well as from a declaration given by the teacher. If the robot, while wandering about the room, learns the location of a particular object, this fact is entered into the semantic store. Thus, because of our restriction that the meaning of sentences must be related to the robot's capabilities, the semantic map is a internal representation of the state of the environment.

The semantic map contains entries for each object in the room, along with the position and description of the object. Since the robot uses grid coordinates to describe position, the obvious data structure for the semantic map is an array the shape of the room. Each entry in the map will contain a list of adjective concepts and a noun concept, thus describing the object located in that position.

3.5.2.2 The Semantic Stack

We have stated that the meaning of a sentence depends upon the individual words comprising the sentence, and the structural relationships between these words. It is the purpose of the semantic stack to represent this structural relationship during the parsing process. When the syntactic parse is completed the semantic stack will consist of a single entry which represents the meaning of the sentence. This entry can be immediately entered in the semantic map.

Each entry on the semantic stack, call a <u>descriptor</u> is capable of expressing a complete "thought" for the robot.

```
┌──────────────────────────────────────────────────────────────────┐
|          |         |       |                |       |       |        |
| PROD #   | START   | END   | ADJ ADJ NOUN   | POS   | ADV   | VERB   |
|          |         |       |                |       |       |        |
└──────────────────────────────────────────────────────────────────┘
     |          |          |                         |
     |          |          |                         └-Position in the
     |          |          |                           room referenced
     |          |          |                           by   this  non-
     |          |          |                           terminal.
     |          |          |
     |          |          └-Index in the sentential form of  the
     |          |            last  symbol  covered  by  this non-
     |          |            terminal.
     |          |
     |          └-Index in the sentential form  of  the  first
     |            symbol covered by this non-terminal.
     |
     └-Index in the grammar of the production used.
```

FIGURE 3.5.2.2

A descriptor is entered onto the semantic stack for each non-terminal in the syntactic parse. As terminal symbols are produced in the parse the corresponding concepts are entered into the descriptor for that non-terminal. The semantic routines modify the descriptors of the non-terminals used in the production in accordance with the concepts represented by the non-terminal. When the parse finishes, the final descriptor is the "meaning" of the sentence symbol of the grammar, and thus the meaning of the entire sentence.

The best way to exemplify the use of the semantic stack, the concept programs, and the semantic routines, is to give a detailed example. The following rules will help explain the example given in the next section.

1) As each production in the grammar _is applied_ an empty descriptor is added to the semantic stack.

2) When a production _is completed_ (when its subtree has been completely reduced to terminals) the associated semantic routine is executed.

3) The semantic routine receives as input the descriptors for the non-terminals on the right hand side of the production.

4) The output of the semantic routine is a new descriptor for the non-terminal on the left-hand side of the production. This new descriptor replaces the empty one currently on the stack for this non-terminal.

5) The semantic routines may call the concept programs in order to alter the descriptors.

NOTE: In the diagrams that follow actual English words appear in the descriptors. The semantic stack is depicted in this way only to make the inner workings more understandable to the reader. The robot deals exclusively with <u>concepts</u>, and thus the real semantic stack contains only numeric codes for the concepts. Also, the stack is shown growing downward, so that the "top" of the stack is at the "bottom" of the picture.

3.5.2.3 Sample Syntactic and Semantic Parse.

As an example of a simple parse using the semantic stack, we will parse the sentence "A table is left-of the chair." The grammar is as follows.

```
  . --> <NP><VP><PP>     (1)
<NP> --> J <NP>          (2)
<NP> --> N               (3)
<VP> --> V               (4)
<PP> --> P <NP>          (5)
```

The semantic routines for each production are as follows.

(1) Store the descriptor elements for <NP> into the semantic map at the location indicated by <PP>.

(2) Change descriptor for <NP> by the concept program for J.

(3) Form a descriptor with concept for N.

(4) Form a descriptor with concept for V.

(5) Change descriptor for <NP> by the concept program for P.

The word-concept map is used to change the words to the goal sentence <u>JNVPJN.</u> The production for the punctuation symbol is applied first.

PRODUCTION APPLIED	SENTENTIAL FORM	#	S	E	SEMANTIC STACK ADJ	ADJ	NOUN	POS	AD	VERB
(1)	<NP><VP><PP>	1	1	3	-	-	-	-	-	-
(2)	J<NP><VP><PP>	1	1	4	-	-	-	-	-	-
		2	1	2	-	-	-	-	-	-
(3)	JN<VP><PP>	1	1	4	-	-	-	-	-	-
		2	1	2	-	-	-	-	-	-
		3	2	2	-	-	-	-	-	-
EXECUTE SEMANTIC ROUTINE		1	1	4	-	-	-	-	-	-
FOR (3)		2	1	2	-	-	-	-	-	-
		3	2	2	-	-	table	-	-	-
EXECUTE SEMANTIC ROUTINE		1	1	4	-	-	-	-	-	-
FOR (2)		2	1	2	-	-	table	0	-	-
(4)	JNV<PP>	1	1	4	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	-
EXECUTE SEMANTIC ROUTINE		1	1	4	-	-	-	-	-	-
FOR (4)		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
(5)	JNVP<NP>	1	1	5	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	5	-	-	-	-	-	-
(2)	JNVPJ<NP>	1	1	6	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	6	-	-	-	-	-	-
		2	5	6	-	-	-	-	-	-
(3)	JNVPJN	1	1	6	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	6	-	-	-	-	-	-
		2	5	6	-	-	-	-	-	-
		3	6	6	-	-	-	-	-	-

PRODUCTION APPLIED	SENTENTIAL FORM	#	S	E	ADJ	ADJ	NOUN	POS	AD	VERB
EXECUTE SEMANTIC ROUTINE FOR(3)		1	1	6	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	6	-	-	-	-	-	-
		2	5	6	-	-	-	-	-	-
		3	6	6			chair	-	-	-
EXECUTE SEMANTIC ROUTINE FOR (2)		1	1	6	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	6	-	-	-	-	-	-
		2	5	6			chair	25	-	-

Note: The concept program for "the" fills in the position of the chair last referred to in the dialog. (25 in the example)

PRODUCTION APPLIED	SENTENTIAL FORM	#	S	E	ADJ	ADJ	NOUN	POS	AD	VERB
EXECUTE SEMANTIC ROUTINE FOR (5)		1	1	6	-	-	-	-	-	-
		2	1	2	-	-	table	0	-	-
		4	3	3	-	-	-	-	-	is
		5	4	6		(empty)		24	-	-

Note: The concept program for "left-of" calculates the coordinates of the position left of the input position.

PRODUCTION APPLIED	SENTENTIAL FORM	#	S	E	ADJ	ADJ	NOUN	POS	AD	VERB
EXECUTE SEMANTIC ROUTINE FOR (1)		1	1	6			table	24	-	is

Note: Actually the transformation rules would be applied at this point. Section 3.5.2.5 deals with this topic in detail.

FIGURE 3.5.2.3

3.5.2.4 Automatic Linkage of Semantic Routines

From the previous example the importance of the semantic routines should be evident. Given a specific context-free grammar it is not hard to define a semantic routine for each production. We simply write a program that performs the desired task for a given production in the grammar. We are faced with a somewhat more complex problem, since the robot must be capable of working with various grammars. The robot must have some way of automatically associating semantic routines to productions. It would be of little value for the robot to infer a grammar by itself only to require human intervention to program a different semantic routine for each production. The robot must be capable of inferring and <u>using</u> the grammar by itself. Therefore, we must give the robot the ability to automatically define the correct semantic routines for any inferred grammar.

Given any grammar inferred in Phase 2, the robot must be capable of defining semantic routines for each production. Furthermore, the semantic routines must provide the desired interpretation of the sentence when it is parsed.

An interesting aspect of this approach to computer linguistics is that this automatic linkage of semantics can be made from the <u>form</u> of the productions. This type of linkage is capable of extracting meaning from complex sentences of the type used in Section 3.5.6. The fact that this is possible is a result of the form of grammars produced by Phase 2 and the descriptor representation of the semantics.

Consider the form of productions produced by the Phase 2 inference technique. Basically the technique infers context-free grammars in Chomsky Normal Form. However, due to the cleaning up process required to increase efficiency, two other forms of productions arise. Thus, only four types of productions can be present in a grammar inferred in phase 2.

<NT> --> T (1)

<NT> --> <NT1> <NT2> (2)

<NT> --> <NT1> (3)

<NT> --> T <NT1> (4)

Where T stands for any terminal and <NT>, <NT1>, and <NT2> stand for any non-terminal.

The first two forms of productions are the Chomsky Normal Form productions. A production of type(3) can occur if a "selection" takes place during the cleaning up process. A production of type(4) occurs only because of the final cleanup in which certain instantiating productions (type(1)) are backed up into type(4) productions in order to speed parsing.

The semantic routines for productions of the form <NT> --> T require the building of a descriptor in which the terminal concept is inserted into the appropriate cell of the descriptor. The cell is determined by the part of speech of the terminal concept. Since an empty descriptor is already on the semantic stack for <NT> at the time the semantic routine is called, it is somewhat more accurate to say that the terminal concept is added to the already existing descriptor as opposed to creating a new one. In any case, the resulting descriptor contains only the concept number for the terminal on the right hand side of the production.

For productions of type(2) <NT> --> <NT1> <NT2> the semantic routine will form a new descriptor from the two existing descriptors on the top of the stack. The two existing descriptors are "combined" into a single descriptor in the following way. The cells of the second descriptor (for <NT2>) take precedence over the cells of the first descriptor. That is, the resulting descriptor will be the same as the descriptor for <NT2> unless a cell is empty in which case the contents of the descriptor for <NT1> are inserted. In cases where a conflict of descriptions of the same object arise a reference to the semantic map is required to combine the two descriptors in a manner consistent with the known state of the environment.

A good example of combining descriptors is the following complex noun phrase "The table which is right of the green chair." The production of interest would be:

<COMPLEX NP> --> <NP> <CLAUSE>

When the semantic routine for this production is executed the semantic stack would already contain descriptors for <NP> and <CLAUSE>. Assuming the last referenced table is in position 25 and a big table is in position 17 and ~~with~~ a green chair in 16, the stack for this noun phrase would be:

<COMPLEX NP>		
<NP>	table	25
<CLAUSE>	big table	17

The semantic routine would ccmbine these into the following descriptor.

<COMPLEX NP>	big table	17

FIGURE 3.5.2.4

It is clear from this example why the second descriptor takes precedence over the first descriptor, since the table referenced by the sentence must be the one in position 17.

The semantic routine for type(2) productions works not only for these restrictive clauses, but also for combining the parts of the predicate, and even for combining the subject and the predicate to form the final descriptor for the sentence. In these latter cases, there is generally no conflict between the the cells and the descriptors can simply be merged.

Chain productions of the form <NT> --> <NT1> (type(3)) are useful in the syntactic analysis of the sentence but play no role in the semantic analysis. The descriptor for <NT> is simply replaced by the descriptor for <NT1>. The semantic routines view one non-terminal the same as another, they are all represented as descriptors, and therefore a chain production simply passes the descriptor from one non-terminal to another. In fact, the semantic section of the complete parsing technique completely ignores chain productions, it doesn't even enter an empty descriptor for <NT> on the stack. In this way no time is wasted by calling the semantic routine for a chain production that would simply pass back the same information given to it. So in this sense no semantic routine is necessary for type(3) productions.

Type(4) productions perhaps require the most interesting semantic routines. The form of the production is <NT> --> T <NT1>. The descriptor for <NT1> is modified by the concept program for the terminal T. Of course, by the time this semantic routine is called, the descriptor for <NT1> is built and the concept for the terminal T is known.

An example of a concept program would be the program for the concept "left-of". This program takes the position cell of the input descriptor and subtracts 1 from it, thus calculating the coordinates of the position <u>left of</u> the known position. The concept program then references the semantic map to find what object is located at this new position and inserts this information into the adjective-noun cell of the descriptor. The result is a new descriptor, different from the input descriptor, but related to it by the "meaning" of the concept.

Thus, the robot can associate to each production of a grammar inferred in Phase 2, one of the three semantic routines built into the robot. This association is made by examining the form of each production. No attempt has been made to show that these semantic routines working in conjunction with the concept progrmas are capable of extracting the desired meaning from any natural language sentence pertaining to the robot's world. Indeed, I'm sure this is not the case. However, this approach to semantics is capable of dealing with complex sentences of the type used in the sample dialog in Section 3.5.6. What we have found is an automatic technique in which a robot can learn to converse in a restricted range of topics using reasonably complex, natural sounding English.

3.5.2.5 Transformations

In any natural language it is possible to convey a given meaning (the "deep structure") using sentences that are structurally very different (the "surface structure") (Chomsky(1965)). Consider the following examples.

A small chair is right-of the piano. (1)
There is a small chair right-of the piano. (2)
Right-of the piano there is a small chair. (3)

Each of these sentences conveys exactly the same meaning, yet their structures are very different. In all the sentences the subject is "a small chair" and the predicate is "(there) is right-of the piano." In the first example the subject and predicate are completely separate. In the second sentence the predicate is split by the subject and in the third sentence the parts of the predicate are reversed and the subject is at the end of the sentence.

A standard context-free grammar is incapable of coping with these structural rearrangements in a uniform manner. In particular, the second sentence, in which the predicate is split by the subject, causes extreme difficulty for context-free grammars. If we insist on parsing the sentence by including "right-of the piano" and "there is" in the same non-terminal representing the predicate, then the sentence cannot be parsed by any standard context-free grammar. This is because the non-terminals of a context-free grammar must generate consecutive terminal symbols.

Since these structural permutations are common in natural language usage, it is desirable to give the robot the capability of dealing with them. We therefore include the following transformational rules for the context-free grammars that will allow the robot to handle these types of sentences in a uniform manner.

We include within all the grammars inferred by Phase 2, the following two productions.

<SENTENCE> --> <SUBJECT> <PREDICATE> (A)

<PREDICATE> --> <VERB PHRASE> <MODIFYING PHRASE> (B)

Along with inferring the explicit productions for the positive sentences, the Phase 2 inference technique also infers allowable transformations for the starting productions. These starting productions are the productions with punctuation as the non-terminal on the left hand side. The right hand side of these productions consists of the allowable permutations of the <SUBJECT> <VERB PHRASE> and <MODIFYING PHRASE>. These permutations, which need not include all three of the non-terminals, are the only transformations allowed for the context-free grammar.

The parsing of a sentence would begin by applying productions (A) and (B) above, in that order. Then these three non-terminals can be rearranged to any of the existing orders contained within the starting productions for the given punctuation of the sentence. The parse now continues in the manner described in the previous section. Before the semantic routines for productions (A) and (B) are executed the descriptors on the semantic stack are permuted back to the "standard" order of <SUBJECT> <PREDICATE> <MODIFYING PHRASE>. At this point, the semantic routines for (A) and (B) are executed in order with the result being a single descriptor representing the non-terminal <SENTENCE>.

At this point we have reduced sentences with different "surface structures" to identical "deep structures". This "deep structure" is represented internally as the final descriptor for the sentence.

To clarify the transformation procedure we will inspect the late phases of the parses for the three example sentences. The sentences would be parsed with the following starting productions.

. --> <NP><VP><MOD> A big chair is right_of the piano.

. --> <VP><NP><MOD> There is a big chair right_of the piano.

. --> <MOD><VP><NP> Right_of the piano there is a big chair.

The semantic stacks would be as follows when the semantic routine for the starting production is called.

big chair	there is	(empty) 25
is	big chair	there is
(empty) 25	(empty) 25	big chair

At this point the stack is permuted back to "standard" order, and the semantic routine for productions (A) & (B) can be executed.

```
                      ┌──────────────────────────────────┐
<SENTENCE>            |                                  |
                      |──────────────────────────────────|
<SUBJECT>             |big chair                         |
                      |──────────────────────────────────|
<PREDICATE>           |                                  |
                      |──────────────────────────────────|
<VERB PHRASE>         |                        (there) is|
                      |──────────────────────────────────|
<MODIFYING PHRASE>    |(empty)             25            |
                      └──────────────────────────────────┘
```

```
                      ┌──────────────────────────────────┐
<SENTENCE>            |                                  |
                      |──────────────────────────────────|
<SUBJECT>             |big chair                         |
                      |──────────────────────────────────|
<PREDICATE>           |(empty)             25  (there) is|
                      └──────────────────────────────────┘
```

```
                      ┌──────────────────────────────────┐
<SENTENCE>            |big chair           25  (there) is|
                      └──────────────────────────────────┘
```

FIGURE 3.5.2.5

3.5.3 Complete Parse Technique

We have discussed the parsing of simple unambiguous sentences in order to make apparent the underlying mechanism of the linguistic system. We now consider the effect of ambiguity on the performance of the parsing technique. We have chosen a general parsing technique specifically tailored to effectively cope with the problems posed by lexical and structural ambiguity. Let us now consider in detail the manner in which the robot copes with these problems.

3.5.3.1 Lexical Ambiguity

Lexical ambiguity is the ambiguity that arises from a word having more than one meaning. Virtually every word in natural languages is capable of expressing more than one meaning in various contexts.

In a typical sentence where many of these words appear together, the list of possible combinations can grow quite large. In a sentence of only ten words, if each word could express only three possible meanings, there would still be 3^{10} possible interpretations. Thus, the problem of lexical ambiguity is more than just finding a algorithm that is theoretically capable of coping with it, we are faced with the problem of finding a procedure for solving a combinatoric problem in a reasonable amount of time.

Within our linguistic framework we have a very simple means of storing multiple meanings for words by allowing a multiple mapping in the word-concept map. A given word can map to any number of concepts, thus representing the multiple meanings of the word. Given this flexible representation for lexical ambiguity, how can we effectively combat this exponential expansion?

Viewed in terms of the heuristic search, these lexical ambiguities represent a number of possible goals for the search. We must modify the search process to search for a goal that is not explicitly known. The naive approach to the problem is to calculate all the possible permutations of word meanings for the given sentence, and then use the heuristic search to look for <u>any</u> of these interpretations. We would reduce the work somewhat by this approach since the parser deals only with parts of speech and some of the words may map to concepts of the same part of speech. However we have not really made any headway against the exponential growth of the possible interpretations. The cost of calculating the heuristic h' would be prohibative since we must compare the sentential form against <u>all</u> the possible interpretations at each step in the parse. For this reason we must abolish this approach since it cannot be a practical technique in view of the number of lexical ambiguities.

The approach that is used to combat this problem is to specify only as much of the goal sentence as is needed at any given time. Since the instantiating productions only lay down one terminal at a time, we need only know this 1 extra symbol in the goal sentence. We simply consider all the possible terminals that the leftmost non-terminal can produce, and compare this to the corresponding part of speech of the concept involved. If the terminals mismatch then the entire parse is dropped. Since we are working left to right, one symbol at a time, the dropping of a single parse eliminates many possible lexical ambiguities.

Consider the ten word sentence with three meanings per word. If we could drop one possible meaning from the first word because the grammar disallows that part of speech, then we have eliminated 3^9 of the 3^{10} possibilities! In this manner we are really breaking down the exponential growth of the possible ambiguities.

One further point requires mentioning. There is some problem caused when a word has multiple meanings that are both the same part of speech. In this case, even though the sentential forms remain identical, the semantic stacks will differ. We simply add another node to the search by placing both sentential forms (and their semantic stacks) on the open list. Thus, we are indeed searching a tree defined as much by semantics and by syntax.

3.5.3.2 Structural Ambiguity

Structural ambiguity is the ambiguity that arises from the different syntactic relationships that can exist between groups of words. In these cases the grammar will allow more than one distinct leftmost derivation of the same sentence. These different derivations will have a marked effect on the meaning of the sentence. An example of a simple phrase that is structually ambiguous is "ordinary language philosphor". Is it the philosopher or the language he studies that is "ordinary"?

Since we have avoided the use of automatic parsing techniques, which cannot cope with an ambiguous grammar, we find that structural ambiguity, while an interesting and often amusing aspect of linguistics, does not present a particularly difficult problem. The heuristic search is capable of considering all the leftmost derivations of the sentence at the same time. In Section 3.5.3.3 we will consider the means by which many of these parses can be discarded using the context of the current dialog.

The problem of structural ambiguity becomes somewhat more complex when dealing with common English discourse. In many cases, ambiguous phrases are used to express the "wrong" meaning. People are apt to ask for a "hot cup of coffee" instead of a "cup of hot coffee". In dealing with robots, the problem of such inconsistencies should be obvious!

3.5.3.3 Disambiguation

We have already discussed one of the means of disambiguation. That is the dropping of a node from the parse if there is a mismatch in the leading terminal symbols of the sentential form and the goal sentence. We have seen how this "syntactic disambiguation", when working symbol by symbol can effectively reduce the permutations caused by lexical ambiguity.

Of the sentential forms that pass the test of syntactic disambiguation we weed out those that contradict or seem unlikely in the current context. This "semantic disambiguation" greatly reduces the number of valid interpretations that arise from structural ambiguity.

"Semantic disambiguation" is implemented by forcing the semantic routines to return a true or false value corresponding to the descriptor's agreement with the semantic map. If a semantic routine for any production in the parse of a sentence yields a false value, then this parse is dropped for semantic reasons. In this manner we are checking that every structural relationship, every word modifying another, makes sense.

The phrase "the table left of the chair" while syntactically correct would fail semantically if there was not a table left of a chair anywhere in the room. Also the phrase "left of position 11" would fail semantically since position 11 is in the corner of the room, and nothing can be left of it. These examples just don't make sense in terms of the current dialog, and thus they would be dropped by the semantic routines.

This semantic disambiguation helps the robot select the correct meaning from an ambiguous sentence. If we used the phrase "the table left of the chair which is green" we would normally be refering to a green chair, but if there were no green chair in the room then the robot would look for a green table positioned left of any chair. If neither of these conditions were present then the entire parse would fail. On the other hand, what if both arrangements of chairs and tables could be found in the room? Which meaning should the robot select? We prefer not to have the robot make an arbitrary decision when two parses make both syntactic and semantic sense. The robot simply replies that the sentence is ambiguous and asks the teacher which meaning was intended. The heuristic search is capable of finding all the interpretations if we continue the search after finding a goal until the open list becomes empty. This must happen eventually for a context-free grammar since the sentential forms keep getting longer, and eventually all of them will be longer than the goal sentence.

Asking for clarification is done only if more than one parse is successfully completed. It is not done at the moment the two parses split. This is done because the ambiguity may be resolved by some later information in the sentence. Quite often a parse that appears to be an ambiguity will fail syntactically at a later point because the length of the sentence is too short or too long. If the robot were to prompt for clarification at the time when two parses seemed possible, he might be asking about a syntactically incorrect interpretation.

Many sentences in English are really ambiguous, yet people generally only extract one meaning. Quillian(1966) gave an example of this with the sentence "he threw the man in the ring." Normally a person would only conjure up one image from this sentence, that of throwing a man into a ring. However, if asked to think of other interpretations, it is easy to think of two or three more. For example, "while in the ring he threw the man", or "of all the men in sight he threw the man standing in the ring." Since it is apparent that people do not always try to find all the ambiguities in every sentence, it seems undesirable for the robot to continually try to parse every sentence all possible ways.

By stopping the heuristic search after finding any valid interpretation, the robot will use the least expensive, syntactically and semantically correct parse. If we operate the parser in this way then the robot will not always try ~~for~~ to find all ambiguities, but select the simplest meaningful interpretation. In this way the robot operates in two modes, one that only tries to find one valid interpretation, and another mode that tries to find all interpretations of the sentence.

3.5.3.4 Sample Complete Parse

As an example of lexical ambiguity we will demonstrate the complete parse technique on the sentence "There is a step over there." We have already indicated that the word "step" can be used as either as a noun or a verb. The word "there" is similarly ambiguous. In most cases it is an adverb, but in other cases it can be a pronoun as in "Move over there!" where "there" is the object of the preposition "over". Thus, for the sentence "There is a step over there," there are 8 lexical interpretations.

A V J N P A
A V J N P N
A V J V P A
A V J V P N
N V J N P A
N V J N P N
N V J V P A
N V J V P N

As mentioned in the text, the complete parse technique never calculates all of these possibilities, as we have done here, but attempts to parse the sentence one word at a time. As each terminal symbol is laid down by the parse, it is checked against the possible lexical interpretations for the associated word in the sentence. In this way, if we could determine that N was an illegal beginning for a sentence, we could eliminate 4 of these interpretations without ever having calculated them!

As we shall see, with the following grammar a sentence can begin with either A or N.

. --> <NP> <VP> <PP>

. --> <VP> <NP> <PP>

. --> <PP> <VP> <NP>

<NP> --> J <NP>

<NP> --> N

<VP> --> A <VB>

<VP> --> V

<VB> --> V

<PP> --> P <NP>

```
                              .
                              .
        .---------------------.---------------------.
        |                     .                     |
   <NP><VP><PP>         <VP><NP><PP>           <PP><VP><NP>
        |                     .                     |
   .----+----.             ...----------.           |
   |         |             .            |           |
 J...*    N<VP><PP>   A<VB><NP><PP>   V...*       P...*
             |             .
       .-----+----.        .
       |          |        .
    NA...*     NV<PP>   AV<NP><PP>
                  |        .
                  |        ..-------------.
                  |        .              |
              NVP...*   AVJ<NP><PP>    AVN...*
                           .
                           .
                           .
                       AVJNP<NP>
                           .
              .------------......
              |                 .
          AVJNPJ...*         AVJNPN
```

* -- Indicates a syntactic failure due to the rightmost terminal being an illegal part of speech for its associated word in the sentence.

FIGURE 3.5.3.4a

Note that the parse never attempted to view "step" as a verb or the second use to "there" as an adverb, since the grammar would not allow these parts of speech at their respective locations.

As an example of the complete parse technique working on a semantically ambiguous sentence we consider the sentence "Move to the table that is left-of the chair which is green!" The normal interpretation of this sentence would be "Move to the table left-of the green chair!" But if we consider for a moment the sentence "Move to the table in position 21 which is green?", which is structurally similar to the first sentence, we must agree that a possible interpretation of the former sentence is "Move to the green table left-of the chair!" Either or both of these interpretations could be ruled out if there isn't the appropriate arrangement of tables and chairs in the room. Therefore, we must demonstrate the capability of the complete parse technique to find both interpretations and check each semantically as well. We will use the following grammar which is in normal form except for the <CLAUSE> productions, which are more visible in the given format.

```
             ! --> <VP> <PP>
         <CNP> --> <NP> <CLAUSE LIST>
 <CLAUSE LIST> --> <CLAUSE><CLAUSE>
          <NP> --> J <NP>
          <NP> --> N
          <VP> --> V
          <PP> --> P <CNP>
      <CLAUSE> --> RV <PP>
      <CLAUSE> --> RVJ
```

The symbol "R" stands for a relative pronoun. Let us assume that there is a green chair at position 24 and a table in position 23 in order to demonstrate the semantic failure of one interpretation.

```
         |
     ----+----
     |       |
   <VP>    <PP>
     |       |
     |  -----+-----
     |  |         |
     |  |       <CNP>
     |  |         |
     |  |    -----+------
     |  |    |          |
     |  |   <NP>     <CLAUSE>
     |  |    |          |
     |  |    |    ------+------
     |  |    |    |           |
     |  |    |    |         <PP>
     |  |    |    |           |
     |  |    |    |      -----+-----
     |  |    |    |      |         |
     |  |    |    |      |       <CNP>
     |  |    |    |      |         |
     |  |    |    |      |    -----+-----
     |  |    |    |      |    |         |
     |  |    |    |      |  <NP>    <CLAUSE>   <===
     |  |    |    |      |    |         |
     V  P   JN   RV      P   JN        RVJ
```

<VP>			move
<PP>			
<CNP>			
<NP>	table	23	
<CLAUSE>			is
<PP>			
<CNP>			
<NP>	green chair	24	
<CLAUSE>	green		.is

FIGURE 3.5.3.4b

```
              |
        ------------
        |          |
      <VP>       <PP>
        |          |
        |   ------------
        |   |          |
        |   |        <CNP>
        |   |          |
        |   |     -------------
        |   |     |           |
        |   |   <NP>     <CLAUSE LIST>
        |   |     |           |
        |   |     |      ----------
        |   |     |  <CLAUSE>   <CLAUSE>   <====
        |   |     |      |          |
        V   P    JN    RVPJN       RVJ
```

<VP>	move
<PP>	
<CNP>	
<NP>	table 23
<CLAUSE LIST>	
<CLAUSE>	table 23 is
<CLAUSE>	green is

FIGURE 3.5.3.4c

We have indicated the parse trees for each interpretation and the semantic stack for each parse at the lowest level (indicated by the arrows). The semantic stack contains a descriptor for each non-terminal above the arrow and the descriptor is filled in for each non-terminal that leads only to terminal symbols. Thus, the stacks are in the position to begin "backing-up" the tree executing the semantic routines.

For the first parse the semantic routine for the production <CNP> --> <NP> <CLAUSE> will try to combine the two descriptors into a single descriptor for <CNP>. This can be done since the chair in position 24 really is green. Therefore this parse will continue in the standard manner until completion.

The semantic routine for the second interpretation will be for the production <CLAUSE LIST> --> <CLAUSE> <CLAUSE>. The combining of the two descriptors <u>fails</u> because the table in position 23 is not green. Thus, this entire interpretation fails, leaving only the previous parse on the open list, and the final meaning of the sentence is clear to the robot.

3.5.4 Effects of Declarations, Questions, and Commands

We have discussed how the robot transforms an input sentence in English into an internal representation containing the meaning of the sentence. This internal form is the final descriptor for the sentence. If the sentence is ambiguous then there may be more than one final descriptor, but we assume the teacher has indicated the one that represents the intended meaning. Given this final descriptor, how does the robot use it to store the meaning, or answer the question, or obey the command?

If the sentence is a declaration the meaning of the sentence must reflect a change in the semantic map or the sentence would have failed semantically. It is clear what should be done to the semantic map when we inspect the final descriptor.

big green tree	17		is

FIGURE 3.5.5.1a

The verb must reference a concept program that has the effect of altering the map. The adjective and noun concepts would simply be inserted in the semantic map at the location indicated by the position cell of the descriptor. If the adverb specified is the negation concept, such an entry would be removed from the map. In either case the concepts contained in the final descriptor are sufficient to store the meaning, and they are also used to begin the formulation of concepts required to construct a reply. We deal with the topic of speaking in Section 3.5.5.

If the sentence is a question, the answer to the question is contained in the final descriptor. If the question is a "what" or "where" question the answer is already in the adjective-noun, or the position cell of the descriptor. And The concept program for the definite, or indefinite article inserts a position and description of the object. Thus the question "where is the green tree?" would have a final descriptor as follows.

big green troe	17	where	is

FIGURE 3.5.5.1b

The big green tree was the last referenced tree. Clearly the answer is contained in the position cell of the descriptor. The effect of the concept program for the adverb "where" is to strip the descriptor of everything but its position element. This is also used in dependent clauses such as: "... left of where the tree is." The result of "where the tree is" is simply a descriptor with the appropriate position element inserted.

If the question is a yes/no question, it has been answered when the modifying phrase of the predicate was compared to the subject. This was done just after the re-transformation of the semantic stack. For the sentence "is the tree green?" the stack would be:

red tree	47		
green			is

When the descriptors are combined by the semantic routine, the adjectives are compared, and since the subject is a complete description of the tree, the answer is clearly no. The negative concept would then be entered into the adverb cell of the final descriptor.

green tree	17	no	is

The semantic routine for combining descriptors (discussed in Section 3.5.2.4) must check for such consistencies throughout the parse, not just at this low level. If the phrase "the tree which is green" was used and no green tree was in the room, the combining semantic routine must signal this fact. This is quite analgous to answering the question about a specific tree being green or not.

If the sentence is a command the response is simply to call the concept program for the verb, passing as a parameter the rest of the final descriptor. The concept program for the verb, which presumably connects to the hardware, will actually cause the action to be carried out. This program can extract from the final descriptor where to move to, or where to throw something. In any case we leave it to the concept program for the verb to carry out the mechanical action, the linguistic system is not concerned with it.

From this description of declarations, questions, and commands, it should be evident that they are treated in a reasonably uniform manner. The required information is always present in the final descriptor, and the concept program for the verb always carries out the desired activity.

3.5.5 Speaking

In most computer linguistic systems the problem of having the computer compose its own responses has been treated lightly compared to the problem of interpreting input sentences. Many such systems have used "fixed-format" output statements composed by the programmer to serve as the speaking device. ELIZA (Weizenbaum(1966)) used reorderings of the input to created the output. Winograd(1971) used a simple grammar for composing replys that were based a great deal on the actual input words.

It is clear that in an adaptive environment where the actual lexicon available to the robot may change, that fixed-format replies are out of the question, since the words in the fixed format reply may not be contained in a different lexicon. In particular, if we wish to use the adaptive language system on different natural languages, then the fixed format replies would be constrained to a single natural language.

The problem of speaking is not given the same treatment as interpretation since it is hard to conceive of how the computer formulates what it is going to say, let alone construct a syntactically and semantically correct sentence. Once again the use of concepts defining the robot's capabilities makes the problem somewhat more tractable. The list of concepts that the robot is currently using define what the robot is currently "thinking" about. When the robot wishes to speak he could use the current grammar to order these concepts in a syntactically correct form and use the inverse word-concept mapping to speak using the current lexicon. Using this as our basic approach, let us look in more detail at each of the processes involved.

3.5.5.1 Internal Formulation of Concepts

Before the robot can speak he must formulate the ideas he wishes to transmit. At various locations in the program where it seems appropriate for the robot to speak, a set of concepts is always available. Let us consider the following times in which we might wish the robot to speak.

1) Responding to a declaration

2) Answering a question

3) Requesting clarification on ambiguities

There is no real need for the robot to respond to a declaration using a full sentence, since we can tell if the robot understood the sentence by inspecting the semantic map. For this reason many linguistic systems (Winograd(1971)) have responded to declarations with "OK." However, this is an excellent place for the robot to construct a full sentence reply to convey his interpretation of the teacher's sentence, and we will use this opportunity to study some of the aspects of speaking.

After the robot has altered the semantic map he still has the concepts contained in the final descriptor with which he can construct a reply, and this is virtually all the robot needs to construct his reply. For the sentence "a large table is at location 25" the final descriptor would be:

large table	25		is

FIGURE 3.5.5.1a

Thus, the robot has the noun phrase, the verb, and the modifier already in mind to begin formulating his reply. To make the sentence pleasant sounding, certain other concepts for "the", "in", and "position" might be needed. By adding these concepts the robot could construct the reply "the large table is at position 25." These other concepts are added because the sentence is formulated in a specialized area of the robot's programs, and this area inherently deals with these concepts.

We find a similar situation when constructing answers to questions, the required concepts are contained in the final descriptor. For yes/no questions the single word answer is in the adverb cell of the final descriptor. From this alone we can answer either "yes" or "no". But there is also other information contained in the final descriptor, and using it the robot could reply "yes, the tree is green." However, this causes a problem in that such a reply might not be allowed by the current grammar. For this reason we stick with the one word reply.

For "what" and "where" questions the final descriptors also contain the concepts required for a response. Again we can choose a full or partial response. In answer to the question "where is the tree?" the robot could answer "position 21" or "the tree is in position 21." In either case the important concepts are available.

When requesting clarification on ambiguities, all interpretations have been completely parsed and thus there are more than one final descriptor, each of which can be used to formulate a reply. An interesting problem here is that quite often the constructed replies will have the identical wording even though they are internally represented quite differently. This may happen because of a lexical ambiguity (which prints the same word for two different concepts) or from a more basic inability of the robot to express the salient features of each interpretation. To get an idea of the complexity involved refer to the example of Section 3.5.3.3 "he threw the man in the ring." The internal representation of all the interpretations would be quite simple and distinct (due to the different concepts used for "in"), but notice the linguistic gymnastics required to express the differences clearly in English!

Thus, we can see how the robot formulates the concepts he wishes to express. The key concepts come from the final descriptor and the the secondary concepts are introduced by the location in the robot's program where the "thinking" is going on. At this point the concepts are still only a vague list of what is to be said, we must still consider how these concepts are changed into a valid sentence.

3.5.5.2 Ordering of Concepts Using the Grammar

Once the concepts to be expressed are formulated they are reordered using the same grammar used for interpretaing sentences. This reordering process takes a good deal more time than the parsing of the identical sentence, thereby making speaking is a more costly operation than interpreting. Although disheartening, this fact should not be surprising since it coincides with human experience. It is much harder to compose sentences in a foreign language than to read them.

Exactly why the speaking process is so difficult for the robot becomes evident when we examine the heuristic search used to reorder the concepts. We use the heuristic search to search the same tree used in parsing, the tree defined by the context-free grammar. But when speaking we are searching for a <u>permutation</u> of the known parts of speech. The reason the search is so much more costly is that we can no longer find as many syntactic failures. Thus, the number of nodes on the open list increases and the entire search requires more time. This is to say that there are more possibilities to consider when speaking, than when listening.

As an example consider the reordering of the following concepts.

"TABLE" "22" "IS" "THE" "IN" "POSITION"

These concepts would be viewed as the following parts of speech NNVJPJ. We will use the following grammar to order them.

<.> --> <NP> <VP> <PP>

<NP> --> J <NP>

<NP> --> N

<VP> --> V

<PP> --> P <NP>

The search must consider all of the nodes of the tree below.

```
                              <.>
                               |
                               |
                         <NP><VP><PP>
                               |
                 +-------------+-----------------+
                 |                               |
          J<NP><VP><PP>                      N<VP><PP>
                 |                               |
        +--------+----------+                    |
        |                   |                    |
  JJ<NP><VP><PP>       JN<VP><PP>             NV<PP>
        |                   |                    |
     +--+---+               |                    |
     |      |               |                    |
     *   JJN<VP><PP>      JNV<PP>              NVP<NP>
            |               |                    |
            |               |                 +--+---+
            |               |                 |      |
        JJNV<PP>         JNVP<NP>         JVPJ<NP>   *
            |               |                 |
            |           +---+---+          +--+------+
            |           |       |          |         |
        JJNVP<NP>   JNVPJ<NP>   *     NVPJJ<NP>      *
            |           |                  |
          +-+-+      +--+--+            +--+--+
          |   |      |     |            |     |
          *   *      *   JNVPJN         *   NVPJJN
```

* denotes a syntactic failure. These nodes are dropped from the search.

FIGURE 3.5.5

Compare this search tree to the one in Section 3.5.1.3. Since we are dealing with permutations instead of a discrete goal, we cannot eliminate nodes as soon as we can while parsing. If we included the transformational rules for a sentence the search tree would be even bigger! We can avoid this extra cost of the search if we take care to originally order the concepts in a more convenient manner. In fact, this is what is done when using English, but the search must be performed for any other language with a different grammar. For example in the French dialog, the concepts were initially in the English order and thus the reordering search had to be performed for every sentence spoken by the robot.

Once the concepts are ordered according ~~the~~ To the grammar, the inverse word-concept map is used to generate the actual words used in the reply. Since the inverse mapping for a given concept may relate to many words (these words would be synonyms) the robot may have a choice or words. In the dialog which follows the choice is to always take the first word listed in the word-concept map. Thus, the use of "in" and not "at", "position" and not "location" even though all the words are used in the sentences given to the robot. This makes it somewhat more evident that the robot deals exclusively with concepts and not with individual words.

Using this technique the robot is capable of improving his speaking capability with time as the lexicon and grammar improve. It is felt that this approach is both more natural and more flexible than the fixed-format and input-feedback techniques generally used in computer linguistic systems.

3.5.6 Sample Dialog - Execution of Phase 3

This section is a sample execution of the APL/360 program that implements the Phase 3 Complete Parse Technique. Each pair of sentences represents a communication between the robot and the teacher. The first sentence is the teacher's declaration, question, or command, which is followed by the robot's response. All sentences preceded by periods (.....) are programmer formatted output, not composed by the robot. These messages are generally given to indicate some hardware interface. One can think of these sentences being replaced by lights on the control panel of the robot. All of the other sentences are composed by the robot as described in Section 3.5.5. The replies early in the dialog appear to be simple restatements of the teacher's sentence, but it will become clear later in the dialog that this is not the means by which the robot speaks.

This entire dialog is preceded by a vocabulary session (Phase 1) as discussed in Section 3.3.2 in order to develop the relationship between the individual words used by the teacher and the concept programs built into the robot. The dialog consists of 5 sessions, each of which is preceded by a grammatical inference session (Phase 2) as appears in Section 3.4.5.

These inference sessions infer a grammar for sentences given by the teacher that are structurally similar to the sentences in the following dialog session. The dialog consists of a conversation using the current grammar, which is improved automatically by the robot between dialog sessions. In this way the structure of the sentences used in the dialog becomes more complex as the dialog progresses.

The indented text is explanatory material and is not part of the dialog between the robot and the teacher. Occasionally in the explanatory text a reference is made to a concept program for a specific word. This is only to make the inner structure of the program more apparent to the reader. The robot deals exclusively with concepts as discussed in Section 3.2, and a specific word may be associated with more than one concept program.

The robot is placed in an 8 by 4 room containing a number of pieces of furniture. The robot is initially unaware of the contents of the room, except for the knowledge of its initial location in position 11. The following two pictures of the room indicate first the true state of the room, and second the robot's knowledge of the room. After each session we will again display the robot's knowledge of the room, indicating its understanding of the teacher's declarations and commands. The dialog begins using the first grammar inferred in Section 3.4.5.

	1	2	3	4
1				
1			BIG	
1			STEP	
2				
2				
2				
3				
3				
3				
4				
4		BIG		
4		CHAIR		
5				
5				
5				
6		BIG		LITTLE
6		SQUARE		SQUARE
6		TABLE		PIANO
7				
7	BIG	LITTLE		LITTLE
7	CHAIR	CHAIR		TABLE
8				
8				
8		PIANO		

	1	2	3	4
1 1 1	ROBOT	UNKNOWN	UNKNOWN	UNKNOWN
2 2 2	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
3 3 3	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
4 4 4	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
5 5 5	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
6 6 6	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
7 7 7	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
8 8 8	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

SESSION 1

A SQUARE TABLE IS AT POSITION 62.
THE SQUARE TABLE IS IN POSITION 62.

The robot can increase its knowledge of the room by a declaration from the teacher. The square table is entered into the robot's semantic map which appears on the second page following. Notice the robot's preference for "in" instead of "at".

A LITTLE CHAIR IS SOUTH_OF THE TABLE.
THE LITTLE CHAIR IS IN POSITION 72.

The concept program for "the" calculates the position of the last mentioned table in the dialog. The concept program for "south-of" calculates the coordinates of the position 1 square south of the table during the combined syntactic and semantic parse. Thus, the robot responds with this numeric description of the location instead of the one used by the teacher.

WHAT IS NORTH_OF THE CHAIR?
THE SQUARE TABLE

The concept program for "what" provides the answer.

NORTH_OF POSITION 23 THERE IS A BIG STEP.
..... SENTENCE DOES NOT FIT CURRENT GRAMMAR.

Since the current grammar cannot parse this sentence it is rejected. The robot will soon increase the grammar to include this sentence.

MOVE TOWARDS IT!
..... MOTOR SWITCHED ON
..... PHYSICAL CONTACT MADE
A BIG CHAIR IS IN POSITION 42.
THE ROBOT IS IN POSITION 32.

The concept program for "it" refers to the last mentioned object in the dialog, in this case the square table. The robot obeys the command by starting to move towards the square table, but bumps into a chair along the way. The robot's hardware observes the object, determining its description which is entered into the semantic map along with the information that the positions along the robot's path are empty. The robot's new position is also entered into the map which would now appear as follows.

	1	2	3	4
1 1 1			UNKNOWN	UNKNOWN
2 2 2	UNKNOWN		UNKNOWN	UNKNOWN
3 3 3	UNKNOWN	ROBOT	UNKNOWN	UNKNCWN
4 4 4	UNKNOWN	BIG CHAIR	UNKNOWN	UNKNOWN
5 5 5	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
6 6 6	UNKNOWN	SQUARE TABLE	UNKNOWN	UNKNOWN
7 7 7	UNKNOWN	LITTLE CHAIR	UNKNOWN	UNKNOWN
8 8 8	UNKNOWN	UNKNOWN	UNKNOWN	UNKNCWN

SESSION 2

The grammar is now changed by the second inference session appearing in Section 3.4.5.

```
NORTH_OF POSITION 23 THERE IS A BIG STEP.
THE BIG STEP IS IN POSITION 13.
```

This sentence is now accepted and understood by the robot. Notice that this sentence requires the transformational rules described in Section 3.5.2.5.

```
STEP TOWARDS THE STEP!
..... MOTOR SWITCHED ON
THE ROBOT IS IN POSITION 32.
```

This sentence contains an ambiguous use of the word "step". Only one lexical ambiguity is allowed by the grammar and thus the robot need not prompt for clarification.

```
THERE IS A SMALL TABLE AT LOCATION 74.
THE LITTLE TABLE IS IN POSITION 74.
```

This is another sentence requiring the transformational rules. Notice the preference for "little", "in", and "position", a result of the ordering in the inverse word-concept map.

```
THE SQUARE TABLE IS BIG.
THE BIG SQUARE TABLE IS IN POSITION 62.
```

The robot's knowledge of the table is changed to include the fact that it is big. Because the concept program for "the" references the semantic map the robot uses the complete description of the table in its reply.

```
WHERE IS IT?
POSITION 62
```

The concept program for "it" provides a description of the last mentioned object and the concept program for "where" uses this description to provide the answer.

QUICKLY WALK OVER THERE!
..... MOTOR SPEED SET TO HIGH
..... MOTOR SWITCHED ON
THE ROBOT IS IN POSITION 63.

The concept program for the last mentioned object is referenced by "there" as well as by "it". The concept program for "quickly" interacts with the hardware by setting the speed of the motor to high.

IS THE TABLE SQUARE?
YES

A comparison of the known description of the table and the given description provides the answer.

IS THE TABLE IN POSITION 74?
NO

"The" table is the last referenced table which is in position 62.

IS A TABLE IN POSITION 74?
YES

"A" table is any table.

The semantic map would be as follows, reflecting both of the robot's walks in the room.

	1	2	3	4
1 1 1			BIG STEP	UNKNOWN
2 2 2	UNKNOWN			UNKNOWN
3 3 3	UNKNOWN			UNKNOWN
4 4 4	UNKNOWN	BIG CHAIR		UNKNOWN
5 5 5	UNKNOWN	UNKNOWN		UNKNCWN
6 6 6	UNKNOWN	BIG SQUARE TABLE	ROBOT	UNKNOWN
7 7 7	UNKNOWN	LITTLE CHAIR	UNKNOWN	LITTLE TABLE
8 8 8	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN

SESSION 3

Once again the grammar is improved by entering new sentences into the Phase 2 machinery. The point of that inference session was to generalize the notion of a noun phrase to include a restricting prepositional phrase. None of the following sentences would have been acceptable to the robot prior to this inference.

IS THE TABLE IN POSITON 62 BIG?
YES

> Notice that the wording of this sentence is virtually identical to the previous two sentences except for the word "big" at the end. Aside from this similarity the structures of the sentences are completely different. The parser had to carry both interpretations until encountering the word "big", at which point the meaning became clear.

SOUTH_OF THE CHAIR IN POSITION 72 THERE IS A PIANO.
THE PIANO IS IN POSITION 82.

> Simply a nested noun phrase which the semantic routines reduce to postion 82.

WALK TOWARDS THE PIANO IN POSITION 82!
..... MOTOR SWITCHED ON
THE ROBOT IS IN POSITION 83.

> This more complex noun phrase can be used anywhere in the grammar.

	1	2	3	4
1 1 1			BIG STEP	UNKNOWN
2 2 2	UNKNOWN			UNKNOWN
3 3 3	UNKNOWN			UNKNOWN
4 4 4	UNKNOWN	BIG CHAIR		UNKNOWN
5 5 5	UNKNOWN	UNKNOWN		UNKNOWN
6 6 6	UNKNOWN	BIG SQUARE TABLE		UNKNOWN
7 7 7	UNKNOWN	LITTLE CHAIR		LITTLE TABLE
8 8 8	UNKNOWN	PIANO	ROBOT	UNKNOWN

SESSION 4

The grammar is now increased to include restrictive dependent clauses by returning to Phase 2 before entering the following sentences.

```
WHERE IS A TABLE THAT IS BIG?
POSITION 62


THERE IS A SQUARE PIANO THAT IS LITTLE NORTH_OF THE TABLE IN
     POSITION 74.
THE LITTLE SQUARE PIANO IS IN POSITION 64.
```

Notice how the robot simplifies the sentence in its reply.

```
WHAT IS SOUTH_OF THE TABLE THAT IS IN POSITION 62?
THE LITTLE CHAIR
```

	1	2	3	4
1				
1			BIG	
1			STEP	UNKNOWN
2				
2				
2	UNKNOWN			UNKNOWN
3				
3				
3	UNKNOWN			UNKNOWN
4				
4		BIG		
4	UNKNOWN	CHAIR		UNKNOWN
5				
5				
5	UNKNOWN	UNKNOWN		UNKNOWN
6		BIG		LITTLE
6		SQUARE		SQUARE
6	UNKNOWN	TABLE		PIANO
7				
7		LITTLE		LITTLE
7	UNKNOWN	CHAIR		TABLE
8				
8				
8	UNKNOWN	PIANO	ROBOT	UNKNOWN

SESSION 5

The grammar is now made to include other dependent clauses.

A BIG CHAIR IS LEFT_OF WHERE THE LITTLE CHAIR IS.
THE BIG CHAIR IS IN POSITION 71.

The concept program for "where" provides the necessary information to the concept program for "left-of", just as it does when called upon to answer a question.

IS WHAT IS AT LOCATION 72 LITTLE?
YES

The concept program for "what" provides the complete description of the object which is then compared with the teacher's description.

STEP TO WHERE THE TABLE THAT IS SMALL IS!
..... MOTOR SWITCHED ON
THE ROBOT IS IN POSITION 84.

A nested usage of the more complex structures.

	1	2	3	4
1				
1			BIG	
1			STEP	UNKNOWN
2				
2				
2	UNKNOWN			UNKNOWN
3				
3				
3	UNKNOWN			UNKNOWN
4				
4		BIG		
4	UNKNOWN	CHAIR		UNKNOWN
5				
5				
5	UNKNOWN	UNKNOWN		UNKNOWN
6		BIG		LITTLE
6		SQUARE		SQUARE
6	UNKNOWN	TABLE		PIANO
7				
7	BIG	LITTLE		LITTLE
7	CHAIR	CHAIR		TABLE
8				
8				
8	UNKNOWN	PIANO		ROBOT

As an example of the robot's ability to disambiguate, we give the following example. The mechanism for this example appears in the text in Section 3.5.3.4. We set up the room as follows.

```
             1          2          3          4


     1
     1               GREEN
     1    ROBOT      TABLE      CHAIR


     2
     2                                     GREEN
     2                          TABLE      CHAIR
```

```
MOVE TO THE TABLE THAT IS LEFT_OF THE CHAIR WHICH IS GREEN!
..... SENTENCE IS AMBIGUOUS, ENTER THE NUMBER OF THE DESIRED
      INTERPRETATION.
1. MOVE TO THE GREEN TABLE IN POSITION 12!
2. MOVE TO THE TABLE IN POSITON 23!

□:
   2
..... MOTOR SWITCHED ON
THE ROBOT IS IN POSITION 22.
```

In this case the "2" is entered by the teacher. Notice that the two interpretations given by the robot are composed _after_ each parse is fully complete, not when it first became evident that more than one parse was possible. If one interpretation was invalid, given the known state of the room, the robot woud not have prompted for clarification, but simply obeyed the only meaningful command.

3.5.7 Dialog using French

It is extremely hard for humans to separate words and concepts when dealing with a single natural language. For this reason it was felt that we should test the program on a different language to insure that there is no inherent dependency on English.

French is not a particulary severe test since it is similar to English in so many ways, but it does force a complete change in the lexicon as well as some major changes in the grammar. The most obvious grammatical change is that some adjectives follow the noun modified and some precede it. Since the semantic map does not store concepts in this manner, this grammatical difference will force the robot to perform the reordering of the concepts in order to speak. This is evident in the dialog that follows because the listing of the semantic map stores the concepts in the "wrong" order, yet the robot speaks using these concepts in the correct order.

	1	2	3	4
1				
1				
1	ROBOT	INCONNUE	INCONNUE	INCONNUE
2		GRANDE		
2		VERTE		
2	INCONNUE	ARBRE	INCONNUE	INCONNUE
3				
3				
3	INCONNUE	INCONNUE	INCONNUE	INCONNUE
4				
4				
4	INCONNUE	INCONNUE	INCONNUE	INCONNUE

QUELLE EST LA GRANDE ARBRE?
LA GRANDE ARBRE VERTE EST EN POSITION 22.

The robot speaks a full sentence reply to indicate the correct ordering of the adjectives. The semantic map shown on the previous page stores the description of the tree in a form convenient for searching. When speaking the robot must use the grammar to correctly order the words.

IL_Y_A UNE PETITE CHAISE ROUGE DROITE_DE LA ARBRE ROUGE.
LA PETITE ARBRE ROUGE EST IN POSITION 23.

the idiom "il-y-a" causes no difficulty because it was mapped to the concept "is" during Phase 1, and the inferred grammar accounts for the different word ordering. In other words, when the robot was given this idiom in the learning sessions (Phases 1 & 2) it simply inferred the vocabulary and the grammar necessary to handle it. Thus, when encountered in the dialog no special attention is required.

MARCHE RAPIDMENT VERS LA CHAISE!
..... MOTOR SPEED SET TO HIGH
..... MOTOR SWITCHED ON
LA ROBOT EST EN POSITION 13.

The French words are now mapping to the built-in concepts for "walk" and "quickly". The programmer-formatted ouput now becomes glaringly evident because it is still in English while the robot responds in French.

	1	2	3	4
1				
1				
1			ROBOT	INCONNUE
2		GRANDE	PETITE	
2		VERTE	ROUGE	
2	INCONNUE	ARBRE	CHAISE	INCONNUE
3				
3				
3	INCONNUE	INCONNUE	INCONNUE	INCONNUE
4				
4				
4	INCONNUE	INCONNUE	INCONNUE	INCONNUE

3.6 Extensions

In this section we discuss possible extensions of the linguistic system that would take further advantage of its concept based design. These extensions are made in the same spirit as the rest of the adaptive system. That is, once the robot is built (programmed), the only means of communication is via the natural language, no further programming is allowed.

3.6.1 Syntactic Extension

We could extend the syntactic complexity of the grammar directly from the dialog with the teacher by simply including the grammar in the semantic map. This means that the grammar can be changed as a result of a declaration just as the knowledge of the room can be changed by a declaration.

This differs from changing the grammar by giving examples of good sentences (positive sentences) and using the automatic inference technique to form a grammar for them. Using this syntactic extension the teacher could directly add, or change productions by typing in a statement that would be parsed by the current grammar, the meaning extracted, and the grammar changed as a result. An example would be a sentence such as the following.

A <COMPLEX NP> is a <NP> followed by a <CLAUSE>.

This sentence would, as a result of its being parsed, add to the grammar the following production.

<COMPLEX NP> --> <NP> <CLAUSE>

In this way the teacher instructs the robot using English. In a sense this justifies the entire adaptive project. Using the automatic inference technique we can develop the basic means of communication necessary to begin describing English using English. Once this basic communication exists we can develop more complex forms of communication using this basic language. Without an initial means of communication it is not clear how one could begin discussing any subject, let alone one as complex as natural language communication.

This syntactic extension puts the teacher in direct control of the learning of the robot, and thus we feel the potential for developing a more complex use of the language is much greater. That is to say that with a human at the wheel controlling the learning process we believe that the robot can progress much further and much faster.

One interesting philosophical aspect of the "bootstrapping" of English is that it forces the teacher to consider the <u>order</u> in which new structures are presented. It will be easier to present more complex structures in terms of existing structures if the groundwork has been properly laid by the teacher.

3.6.2 Semantic Extension

The semantic extension we propose is the defining of new concepts in terms of old concepts. In this way it would be possible to extend the topics of discourse after the robot is built. Defining a new concept means that the appropriate new concept program must be written by the robot itself. This is not impractical if we restrict ourselves to definitions of the following type.

"Shelter" means house or tree.
"Orange" means red and yellow.

In each case the word in quotes is added to the lexicon and mapped to a new concept program which is added to the robot's capabilities. This new concept program is written by the concept program for "means". In each case this new concept program is basically a number of references to the old existing programs for the concepts that comprise the rest of the definition. The programs for the logical connectives ("and","or") would define the logic used within this new concept program for "shelter" or "orange".

This capability of defining new concepts allows the robot to perform simple inference problems without resorting to theorem proving. Using our definition of "orange" we could then ask the robot:

Is the table orange?

Since the information supplied by the hardware about the table is only in terms of red and yellow (the original basic concepts) it requires simple inference to determine whether the table is orange. This simple inference is identical to Winograd's(1971) inference using the definition of "nice". The reason that the robot is capable of performing this inference without using theorem proving is that a complete description of the object is contained in the semantic map. The concept program for "orange" simply checks for the occurrence of "red" and "yellow" in this complete description of the object.

Again an interesting philosophical question results from extending the system. Given this capability to define new concepts in terms of old concepts, does there exist a <u>basic set of concepts</u> that can be used to define a working set of concepts that would be sufficient to deal with a substantial subset of a natural language?

3.6.3 Semantic vs. Syntactic Complexity

As we increase the complexity of the linguistic system both the grammar and the semantic routines will be called upon to perform more detailed work. In this section we discuss some of the problems that arise as the language used by the robot becomes more complex. After defining the specific problems encountered, we must try to decide whether the grammar or the semantic routines can more naturally cope with the problems.

The first problem is that of word agreement. In natural languages the subject must agree with the verb in person, number, and gender. Similarly, agreement must be made by adjectives modifying nouns, etc. This problem is ~~always~~ often stated as a reason why context-free grammars cannot cope with natural languages (Winograd(1971)). The argument is that the only way such a grammar can handle word agreement is to replicate the entire grammar for each such combination. That is, each production of the grammar is copied using new non-terminals that reflect the person, number, and gender of the subject and predicate. Thus a simple production

<SENTENCE> --> <SUBJECT> <PREDICATE>

could be made to include agreement in number by changing it to the following productions.

<SENTENCE> --> <SINGULAR SUBJECT> <SINGULAR PREDICATE>

<SENTENCE> --> <PLURAL SUBJECT> <PLURAL PREDICATE>

All the productions for <SUBJECT> and <PREDICATE> (not shown) must also be replicated using the new non-terminals introduced. Thus, it is clear that even a simple grammar for English would require hundreds of productions (that are basically the same) just to handle agreement of this sort.

This argument is very convincing indeed, however it overlooks the fact that the semantic routines for a context-free grammar are ideally suited for such checking. When the semantic routine for the production

<SENTENCE> --> <SUBJECT> <PREDICATE>

is executed, there are only two descriptors on the semantic stack, one for the subject and one for the predicate. Since these descriptors contain the necessary information for checking agreement the check can be made at this time quite easily by the semantic routine. The most important feature is that at this time these descriptors are adjacent to each other on top of the semantic stack independent of how far apart the actual subject and verb happen to be in the sentence. This is the case because the parser reduces the sentence to just these two non-terminals by the time the semantic routine is invoked.

By allowing the semantic routines to check agreement, which incidentally can be done in an analgous fashion for adjective-noun agreement etc., we can maintain the original, simple context-free grammar. Sentences that do not have the correct subject-verb agreement will fail semantically, not syntactically, under this formulation.

Another major problem in dealing with more complex subsets of English is the proliferation of syntactic categories. For example, consider the pronoun "she". It cannot be modified by an adjective in common usage. How can we have the robot detect this?

Once again, most suggestions have been along the lines of checking for this syntactically, thereby making the grammar more complex. The approach is to simply add another part of speech (or syntactic category) to the terminal alphabet. Therefore, the productions for nouns will not affect the use of pronouns, and separate rules can be devised.

But this means we must duplicate the productions that do indicate the similariies between nouns and pronouns. For this reason it may be better to try to include these irregularities into the semantic routines. This could be done by defining a "property list" for each concept. Then, whenever one concept modifies another their property lists are compared by the semantic routines. If there is a mismatch, then the parse would fail semantically. This also would allow keeping the original, simple grammar in which nouns and pronouns are treated similarly.

Actually the differences in approach are not really very dissimilar. The property list for each concept can be thought of as a subscript on the class of a particular part of speech, which is analgous to creating a new syntactic category. The difference is that by including the check mechanism in the semantic routines we are not restricting the manner in which we carry out the check to the very rigid rules of a context-free grammar. We can check special cases using the full capability of the programming language which is computationally more powerful than the context-free grammar.

In this way we are using the grammar to indicate the general structures that <u>do</u> exist for a natural language and imbedding the large number of special cases into the semantic routines where we have the computational power to deal with them.

4. Conclusion

How successful have we been in our development of Adaptive Problem Solving? The model has been shown to be general enough to encompass many past learning systems, but more than that it provides a mechanism for isolating the salient features of each of these works to help discern their good and bad aspects. The model has also been shown to clearly represent the critical problems associated with computer learning, and thus helps the future researcher avoid the pitfalls encountered in previous research.

For each of the important learning processes we have attempted to find suitable algorithms for performing them. To this end we have specified theoretically well-founded algorithms to perform these tasks. We have been sensitive to the practical considerations involved in implementing these algorithms. These practical considerations have led us to the development of the Bandwidth Heuristic Search, a meaningful extension of the heurisic tree search technique. ~~The~~ Bandwidth Heuristic Search for MIN/MAX Trees ~~has been shown to be~~ superior to the alpha-beta minimax procedure ~~for a~~ number ~~of reasons~~.

Finally, after using the adaptive framework to identify the critical problems involved in a learning system, and to provide insight as to coping with these problems, we have applied it to the Natural Language Acquisition Problem. Because of our adaptive approach to this problem, we have viewed it from an extremely basic point of view that limits our use of a priori knowledge or ad hoc tricks. The result is a language-independent learning program that is more readily extendible, in terms of both syntax and semantics, than other linguistic systems. This basic viewpoint also leads to methods for specifying certain complex capabilities for an artificial intelligence, such as the formulation of concepts for speaking, or performing common sense inference problems without requiring theorem proving techniques.

The application to natural language acquisiton serves as an example of a successful encounter with the critical problems involved in learning as formulated by the adaptive model. The linguistic system also serves as an example of a successful learning program in which the self-improvement, rather than yielding only minor improvements, provides the primary basis for successful performance. The program's entire knowledge of the individual words and the grammatical structures of the language are developed by the program itself. (15)

Thus, we feel that an adaptive approach to problem solving is desirable because it provides a fresh approach to the problem, and feasible because of the existence of effective procedures for performing the complex processes involved. For these reasons we hope to see successful applications of this theory to a variety of interesting problems.

5. Bibliography

Bobrow(1966)
Bobrow, D. G., "Problems in Natural Language Communication with Computers", Scientific Report No. 5, Bolt, Beranek and Newman, Inc.

Bobrow(1967)
Bobrow, D.G., Fraser, J.B., Quillian, M.R., "Survey of Automated Language Processing", Scientific Report No. 7, Bolt, Beranek and Newman, Inc.

Bush(1955)
Bush, R.R. and Mostello, F., Stochastic Models for Learning, New York: Wiley

Cavicchio(1970)
Cavicchio, D.J., "Adaptive Search Using Simulated Evolution", Technical Report 03296-4-T, Computer and Communication Sciences Dept., University of Michigan.

Chomsky(1965)
Chomsky, N., Aspects of the Theory of Syntax, M.I.T. Press, Cambridge, Mass.

DeRemer(1969)
DeRemer, F.L., "Practical Translators for LR(k) Languages", Project Mac Technical Report MAC-TR-65, Massachusetts Institute of Technology

Fiegenbaum(1963)
Fiegenbaum, E.A., Feldman, J., Computers and Thought, New York: McGraw

Feldman(1969)
Feldman, J.A., Gips, J., Horning, J.J., Reder, S., "Grammatical Complexity and Inference", Technical Report CS125, Computer Science Department, Stanford University

Feldman(1970)
Feldman, J.A., "Some Decidability Results on Grammatical Inference and Complexity, Stanford Artificial Intelligence Project Memo AIM-93.1, Stanford University

Fodor(1964)
Fodor, J.A., Katz, J., "The Structure of a Semantic Theory", _The Structure of Language,_ Prentice Hall, Englewood Cliffs, N.J.

Fogel(1966)
Fogel, L.J., Owen A.J., and Walsh, M.F., _Artificial Intelligence Through Simulated Evolution,_ New York: Wiley

Friedberg(1959)
Friedberg, R.M., Dunham, B., and North, J.H., "A Learning Machine, Part II, _IBM Journal,_ June, v. 3, pp 282-287

Friedberg(1958)
Friedberg, R.M., "A Learn&ng Machine, Part I, _IBM Journal,_ January, v. 2, pp 2-13

Gries(1971)
Gries, D., _Compiler Construction for Digital Computers,_ New York: Wiley

Hollstein(1971)
Hollstein, R.B., "Artificial Genetic Adaptation in Computer Control Systems", Technical Report 032960-14-T, Computer and Communication Sciences Dept., University of Michigan

Klopf(1965)
Klopf, A.H., "Evolutionary Pattern Recognition Systems," Report from Bioengineering Section, Information Engineering Department, The University of Illinois, Chicago

Hart(1968)
Hart, P.E., Nilsson, N.J., Raphael, B., "A Formal Basic for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions on Systems Sciences and Cybernetics, Vol. SSC-4, No. 2, pp 100-107

Holland(1967)
Holland, J.H., "Non-linear Environments Permitting Efficient Adaptation", _Computer and Information Sciences - II,_ New York: Academic Press Inc.

Holland(1969)

Holland, J.H., "Adaptive Plans Optimal for Payoff-only Environments," in Proceedings of the Second Hawaii Conference on System Sciences

page

Hopcroft(1969)
Hopcroft, J.E., Ullman, J.D., Formal Languages and their Relation to Automata, Reading, Massachusetts: Addison - Wesley

Horning(1969)
Horning, J.J., "A Study of Grammatical Inference", Technical Report CS 139, Computer Science Dept., Stanford University

Lin(1972)
Lin, S. and Kernigham, B.W., "A Heuristic Algorithm for the Traveling Salesman Problem", Bell Telephone Laboratories Computer Science Technical Report #1, April 10, 1972

Lindsay(1968)
Lindsay, R.K., "Artificial Evolution of Intelligence", Contemporary Psychology, 13, No. 3, March

Little(1963)
Little, J.D., Murty, K.G., Sweeney, D.W., and Karel, C., "An algorithm for the Traveling Salesman Problem, Operations Research, Nov. - Dec. 1963, v. 11, No. 6, pp 972-989

Minsky(1961)
Minsky, M.L., "Steps Towards Artificial Intelligence", Proceedings of the Institute of Radio Engineers, January, 1961, v. 49, pp 8-30. (reprinted in Fiegenbaum(1963))

Nilsson(1971)
Nilsson, N.J., Problem Solving Methods in Artificial Intelligence, New York: McGraw - Hill

Pohl(1970)
Pohl, I. "Heuristic Search Viewed as Path Finding in a Graph", Artificial Intelligence I, American Elsevier

Quillian(1966)
Quillian, M.R., "Semantic Memory", Ph. D. Thesis, Psychology Dept., Carnegie Institute of Technology

no pay

Samuel(1967)
Samuel, A.L. "Some Studies in Machine Learning using the Game of Checkers II - Recent Progress", IBM Journal, November, v. 11, No. 6, pp 601-617

Samuel(1959)
Samuel, A.L., "Some Studies in Machine Learning using the Game of Checkers", IBM Journal, v. 3, No. 3, pp 210-227. (reprinted in Feigenbaum(1963)).

Schank(1968)
Schank, R.C., "A Notion of Linguistic Concept: a Prelude to Mechanical Translators", AI Memo-75, Computer Science Department, Stanford University

Skinner(1953)
Skinner, B.F., Science and Human Behavior, New York: Macmillan

Slagle(1970)
Slagle, J.R., Formal Systems and Non-numerical Problem Solving by Computers, pp 246-273, Berlin: Springer - Verlag

Slagle(1963)
Slagle, J.R., "Game Trees, M & N minimaxing, and the M & N alpha beta Procedure", Lawrence Radiation Laboratory AI Report No. 3, November, Livermore, California

Weizenbaum(1966)
Weizenbaum, J., "ELIZA -- A Computer Program for the Study of Natural Language Communication Between Man and Machine", CACM, January 1966, pp 36-45

Winograd(1971)
Winograd, T., "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language", Project Mac Technical Report MAC TR-84, Massachusetts Institute of Technology.