MARLA is a collection of FORTRAN routines which implements the Shannon-type chess program with alpha-beta cutoffs occurring dynamically. Board positions are updated incrementally. Also involved in position analysis is a production system which models the human chess player's advice-taking, theme, and chess learning in a general sense. Interfacing these two sections, one algorithmic and one semantic, are two attributed translation grammars. Both use a terminal saturation technique to achieve top-down LL(1) parsing. One grammar translates feature descriptions into feature recognizers, the other recognizes and decodes strategies. The program also exhibits some natural language understanding.
MARLA: A Chess Playing Program
by
Andrew Lloyd Rood

A THESIS
submitted to
Oregon State University

in partial fulfillment of
the requirements for the
degree of
Master of Science
Completed May 6, 1977
Commencement June 1978
APPROVED:

Redacted for Privacy

Assistant Professor of Computer Science

Redacted for Privacy

Chairman of the Department of Computer Science

Redacted for Privacy

Dean of the Graduate School

Date thesis is presented May 6, 1977

Typed by Andrew L. Rood for Andrew L. Rood
# TABLE OF CONTENTS

I. Introduction ................................................. 1
   A. Related Studies ......................................... 5
   B. Microscopic analysis .................................... 7
   C. Human Chess Analysis ................................... 9
   D. A Multiphase System .................................... 13

II. A Rudimentary Chess Player ................................. 17
    A. Implementation Section ................................. 17
    B. Control Section ....................................... 23
    C. MARLA .................................................. 25

III. Features and Strategies .................................. 30
    A. Features .............................................. 31
    B. Strategy in MARLA's perspective ....................... 44

IV. The Production System ..................................... 51
    A. Production System Operation .......................... 52
    B. Lines of Play in Chess ................................ 54
    C. Operational Enhancements .............................. 55
    D. Training mode ........................................ 59
    E. Cross Production System Communication ............... 61

V. Conclusion .................................................. 63

Bibliography .................................................. 65
### LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Time Perspective</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>An Example Position</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>Chess Player Block Diagram</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Call Depth</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Data Structures</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Implementation Section</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>Control Section</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>MARLA</td>
<td>27</td>
</tr>
<tr>
<td>9</td>
<td>Grammar Defining Feature Description Language</td>
<td>34</td>
</tr>
<tr>
<td>10</td>
<td>Terminal Symbol Translation Table</td>
<td>38</td>
</tr>
<tr>
<td>11</td>
<td>Partial Parse Tree for Feature: Self-Controlled Center</td>
<td>43</td>
</tr>
<tr>
<td>12</td>
<td>Strategy Defining Grammar</td>
<td>46</td>
</tr>
<tr>
<td>13</td>
<td>Parse Tree for Example Strategy</td>
<td>49</td>
</tr>
<tr>
<td>14</td>
<td>Learning Mechanism Structures</td>
<td>60</td>
</tr>
</tbody>
</table>
MARLA: A CHESS PLAYING PROGRAM

I. INTRODUCTION

Recent advances in computer technology and in software design have brought us, in a practical sense, to the general question of the computer representation of knowledge. In particular, developments in computer list-processing capabilities (such as LISP) have raised the question: should the computer representation of knowledge be primarily algorithmic or primarily semantic? The first step in answering this question involves analyzing the definitions of 'algorithmic' and 'semantic'.

The theoretical basis for algorithmic representation of knowledge can be found in the Turing machine. Just as a Turing machine involves transfer of control to a state which itself explicitly defines the next state and the operation to be performed; so any abstractly algorithmic approach will contain procedures which accept control, possible change the data base, then explicitly transfer control to another procedure. Definitionally, any algorithmic representation must involve procedures and
explicit transfer of control.

The basis for the semantic representation runs back through the study of formal grammars to the production systems of E. L. Post (31). In this type of representation the basic operational unit is a production. A production is a pair of strings, the second of which is used to replace an occurrence of the first in the database. For any particular problem each symbol in the database or in a production has a particular meaning. It is through these meanings that the production system can be used to do meaningful computation. Thus, a production system is by nature semantic. Productions are activated by the configuration of the system's current string (its database) and cause a modification of that string. There is no explicit transfer of control involved.

From a theoretical point of view the two schema are equally powerful (see proof in 31). In practice however, the types of problems usually attacked by the two approaches are quite different. Algorithmic solutions (20,30) usually involve small, complete problems. When algorithmic methods are used to solve large problems (for example: our current operating systems), the results are usually an arbitrary collection of solutions, some ad hoc, which work. Production systems on the other hand are generally
used to attack substantial, often open ended problems (8,10, 12,28). Perhaps the most notable production system application is E. H. Shortliffe's MYCIN, a medical consultation system (36,37) which uses productions to help generate medical diagnoses and treatments.

Extending the question of algorithmic and semantic representation of knowledge to human cognition, one might ask whether human understanding is semantic or algorithmic? This paper makes the assumption that human understanding is both algorithmic and semantic. Its thesis is that practical machine representation of knowledge must use both algorithmic and semantic approaches. In support of this thesis the following chapters describe a chess playing program which embodies both algorithmic and semantic information. This embodiment includes mechanisms whereby the program can learn from a chess instructor (take advice), and examine the games of human chess masters to refine its semantic information and improve its play¹. The program is

¹ Not unexpectedly (if these claims are true) the mechanisms have some relevancy to the problem of natural language understanding. I firmly believe that in order to understand a discipline verbally a program must have some structural preconceptions about that discipline. That is: natural language understanding must stem from some deeper understanding of the structure the natural language describes. If this program can learn to play better and better chess then it must in some primitive sense understand chess. But
effectively a model of the human chess player.

There are several strong reasons why chess is a profitable example to study. Chess, first of all, is a representative of a more general class of perfect information problems which rely on tree searching as a central technique. Chess is a well defined, complete problem with a theoretical solution, yet one complex enough to have resisted all attempts to create a complete practical solution. Furthermore, work has already been done on chess playing programs attempting, with little success, to apply either current algorithmic or current semantic techniques to the problem.

In the remainder of this chapter are a brief explanation of why neither algorithmic nor semantic approaches have solved chess, and the introduction of a general structure wherein the two types of organization can coexist usefully. Additionally, a particular implementation of this system will be outlined and a particular example within that implementation begun. Before continuing however, here is a brief summary of other computer chess efforts.

then it should be able to talk about chess in any language capable of expressing that understanding. During my work this seemed more and more to be the case. MARLA can talk. In this vein I shall point out natural language ramifications of this work as they become appropriate.
A. RELATED STUDIES

The dream of a mechanical chess player which could outplay humans is centuries old. However, not until the advent of the high speed digital computer has such a machine been a viable possibility. Several lucid descriptions of this history have appeared in the literature (19, 36). Beyond these, the literature on computer chess is quite rich, including two bound volumes (7, 23) dealing with this subject alone. Two papers deserve special attention; the first by C. E. Shannon (32, see also 33), and the second by R. D. Greenblatt and others (15). Shannon's paper was the first to suggest applying the min-max tree search algorithm to chess.

The min-max algorithm is a tree searching technique based upon an evaluation function. A chess tree has nodes corresponding to board positions and arcs corresponding to moves. An evaluation function associates a number with each board position. The min-max technique assumes that one player wishes to minimize this evaluation function and that the other player wishes to maximize it. Given any chess tree with evaluation function values known for all and only the leaves (terminal nodes) of that tree, the min-max algorithm by repeated minimization and maximization
associates a value with each non-terminal node and with each arc in the tree. Thus, the algorithm can be used to choose, for any position, which of the moves allowed in that position has the highest (or lowest) value.

R. D. Greenblatt's program also uses the min-max technique. Greenblatt's paper describes the first program competitive on a human level and it will be considered typical of the several other algorithmic, so called Shannon strategy programs (5,13,21). The only major work which systematically augments this min-max strategy is that of A. L. Zobrist (40,41). He has approached the problem from a primitive feature recognition perspective. His work attempts to deal with a chess position in chunks larger than a single piece or square, using analysis of these chunks to direct the tree construction.

These programs are written in many different languages, from LISP to assembler. Greenblatt's is written in FORTRAN. The system described here is implemented in FORTRAN, primarily for the language's efficiency. All of these programs, except perhaps Zobrist's are essentially algorithmic. Unfortunately, after much effort by dedicated groups these programs play at best at an average club or poor tournament level.
B. MICROSCOPIC ANALYSIS

Why is it that this primarily algorithmic approach fails? In a general theoretical sense the algorithmic approach is doomed to mediocrity because the chess tree (which has on the order of $10^{43}$ distinct root to leaf paths) is effectively infinite. That is: the tree is finite, but so large that for practical purposes it is infinite. Consider a Turing machine sitting on this chess tree. The Turing machine cannot succeed in examining the by crawling from node to node because the tree is too large. To do so would require effectively infinite time. Yet, the Turing machine cannot expect the entire tree to be stored for it to jump around on, because the tree is too large. To do so would require effectively infinite space.

Consider this algorithmic, tree searching approach mathematically in terms of individual node evaluation. Assume it takes $N$ seconds to evaluate a node and choose which of its successors to put in the tree. Assume further that we are interested in high level tournament play and therefore a minimum look-ahead of eight moves or 16 half-moves (plies). At tree level 15 the number of nodes will be $A^{15}$ if each parent has $A$ descendants. This level will
evaluate in $A^{15N}$ seconds.

A general process for selecting each A descendants is to sort all possible descendants and select A from the top of this sorted list. Suppose we wish to reduce A by improving the quality of our sort at each node. How much additional time can justifiably be spent at each node to reduce A by one-half? In a tree of width $A/2$ the number of nodes at level 15 is

$$\frac{A^{15}}{2^{15}} = A^{15}2^{-15} < A^{15}10^{-4}$$

If our node evaluation time is increased by X seconds then the 15th level will take less than $A^{15}(N+X)10^{-4}$ seconds to evaluate. However:

$$A^{15}10^{-4}N + A^{15}10^{-4}X = A^{15}N$$

implies

$$10^{-4}X = N - N10^{-4}$$

$$X = (N-N10^{-4})10^4$$

$$X > N10^4$$

If we spent four seconds per node before improvement then we can justify spending 40,000 seconds if we can achieve a 50% reduction in A! The sheer mathematics of the situation demand that we put more effort into individual node analysis (possible semantic analysis).

Can an entirely semantic approach succeed in
solving chess? Our thesis is that a totally semantic approach will not succeed in practice. The difficulty is that the board representation, the system for determining legal moves, the board evaluation function, and other aspects of chess all seem to be inherently algorithmic. This is such an immediate problem that few attempts to implement the rudiments of chess in a non-algorithmic manner have been made. One notable attempt to develop a list-oriented representation (3) is a seventy page tract devoted entirely to mating combinations.

What is it about the board that makes its analysis essentially algorithmic? The fact that the board is a two-dimensional matrix such that many geometric relationships (e.g. in the same row, on the same diagonal, in the center) are simultaneously significant is a key. If the board is decomposed into sets of property lists, then in order to recapture just the board information, each of these relationships must be represented. In an abstract sense, the board contains information most easily gleaned by crawling around on it.

C. HUMAN CHESS ANALYSIS

Before outlining any system let us examine the
processes involved in a human chess master's position evaluation. It has been shown \((45)\) that excellent players will recognize certain key features of any board position. They have learned that certain configurations of pieces on certain parts of the board are significant and with those configurations, or features, they now associate certain strategies. In fact the literature of chess is full of feature definitions and maxims on what actions are implied by those features. It is presumed that some time in the past the chess master has learned:

1. the rules and moves,
2. how to recognize features,
3. which strategies to associate with those features, and
4. how to implement those strategies.

Furthermore, the master has a well defined sense of:

5. how the different strategies interact.

To these five capabilities the chess master adds some tree searching and control capabilities, and he is ready to play. The processes involved in his play include:

1. recognition of features,
2. application of maxims to obtain strategies, and
3. implementation of strategies.

Figure 1 shows a time line indicating when each
CHESS MASTER

1. learn rules
2. learn features
3. learn strategies
4. learn strategy implementation
5. learn strategy interaction
6. recognize features
7. apply maxims
8. implement strategies

COMPUTER CHESS PROGRAM

Figure 1. The Time Perspective
of the above capabilities is acquired and when each process is active.

A general method for combining algorithmic and semantic approaches must include a set of procedures, a set of productions, some mechanism which can provide communication between these two sets, and a control structure to monitor the interaction of the two sets. This control structure can be embedded in the procedure set if one is willing to allow the production system to run to termination whenever it is called (since the algorithmic and semantic approaches involve different control mechanisms it is assumed that they operate in a mutually exclusive fashion). The operation of this type of algorithmically controlled model will be two phase. The first (algorithmic) phase will determine when second phase (production system) computation is required and initiate it. The second phase will always run to completion, its final act always being the restarting of phase one computation.

Formal grammars can be used to solve the communication problem. These grammars are essential to the integration of semantic and algorithmic approaches. Attributed translation grammars can be used to attach meaning to each of the production system symbols. A particular string in the language defined by one of these
grammars will correspond to a particular symbol and will be that symbol's definition. The translation of any particular string in the language will be a procedure which links the symbol's meaning with the algorithmic section. The grammatical computation involved in this production system - algorithm interface can be viewed as a third phase of machine operation.

D. A MULTIPHASE SYSTEM

The program described herein, MARLA\textsuperscript{2}, is based on a rudimentary implementation of the Shannon tree search strategy. All positions, except the initial position, are generated incrementally from previous positions. The terminal node evaluation function considers the number and strengths of the current pieces and the mobility of each. At each node during the tree search the moves are sorted (using evaluation function values) by the routine SORT, and selected by the routine SEARCH. This rudimentary program was prepared for the above chess player model by removing SORT and creating a more complex production system oriented sorting mechanism.

\textsuperscript{2}MARLA- Multiphase Algorithmic, Recursive, and Linguistic Automaton.
MARLA uses an algorithmic approach to move implementation and tree searching. The program uses semantic information in the form of chess maxims at each node in the search tree's construction. The maxims interact in the context of a production system. The program uses context-free grammars to achieve communication between these sections. The grammars are described in Chapter 3, as is the technique used in their development. The production system itself is described in Chapter 4. The actual chess model now consists of an algorithmic section, a semantic (production system) section, a strategy translation grammar, and a feature translation grammar.

MARLA models each of the eight cognitive processes indicated above. The time line of Figure 1 shows the program's activities and their time perspective. The moves and rules are known to the program a priori. Prior to play the program must:

1. accept feature descriptions and produce corresponding feature recognizers,

2. accept strategy descriptions and generate codes for strategy implementation,

3. accept chess maxims, and

4. adjust maxim weights (by examining games) to achieve optimal maxim interaction.
At each node of the tree during play the program will:

1. use the feature recognizers to extract the important features from the current position,

2. use the production system to allow the maxims to come into play and choose strategies, and

3. choose moves which implement the strategies chosen.

Throughout the rest of this thesis the following chess maxim will be considered (see Figure 2):

\[
\text{IF } \begin{align*}
\text{you control the center} \\
\text{and you have a mobile pawn wing}
\end{align*} \text{ THEN } \begin{align*}
\text{advance on that wing with pawns.}
\end{align*}
\]

MARLA is taught to recognize a controlled center and a mobile pawn wing, and is taught to implement the strategy "advance on the wing with pawns". During play the program puts these three together to implement this maxim.
In the above position white controls the center (the outlined squares) and his left pawn wing is mobile (since it has only empty squares ahead of it). Applying the maxim for white will suggest that one of the pawn advances (indicated by the arrows) on the left wing should be made.

Figure 2. An Example Position
II. A RUDIMENTARY CHESS PLAYER

The FORTRAN main program T1, its two related functions, and its 26 related subroutines are a chess playing complex of the Shannon type. The complex uses typical numerical representations: one and two dimensional arrays, a LIFO stack, and a typical doubly linked tree structure.

The functions required of any game playing program are three-fold. First, the program must maintain game status information. Second, the program must interact coherently and responsively with the user. Third, and most difficult, the program must be able to select its own legal moves. The chess playing complex is roughly organized into three parts (see Figure 3) to fulfill these three functions.

In order to better understand the possible calling sequences involved, Figure 4 shows relative procedure depths.

A. IMPLEMENTATION SECTION

Figure 5 shows the four major data structures of the implementation section. The first is a typical board
Figure 3. Chess Player Block Diagram
Figure 4. Call Depths
Figure 5. Data Structures
matrix, IBD(8,8). The second, MOVES(32,30), contains explicit lists of the currently available moves for each piece. The third is a vector of status flags concerning nearly all system functions. The fourth is the game history, a vector of past moves.

Essential to any attempt to implement the game of chess are the individual piece's move generators. In chess there are six distinct piece types, hence six generators: PAWN, NITE, BHOP, ROOK, QUEEN, and KING. These all work by adding sets of index increments to piece positions, detecting moves out of range, and consulting CHECK (see Figure 6). The subroutine CHECK examines IBD and determines whether or not the desired color is in check. CHECK works by adding preset increments to the king position and examining for attackers. The move generators return a left justified list of all and only the moves available. For convenience these six subroutines are all accessed through the multiplexer CHESSM.

Recomputing the moves for all pieces on the board after each player's turn would be quite inefficient since only a fraction of the total number of board pieces are affected by any given move. For any given position, the subroutine AFFLIST attempts to find those and only those pieces which have been affected by the most recent move.
Figure 6. Implementation Section
If a move causes check then all of a player's pieces are affected. AFFLIST considers this. If a player was in check and has just relieved that check then all of his pieces' move potentials must be recomputed. AFFLIST considers this as well. Similarly to the move generators and CHECK, AFFLIST uses a reset-increment method of determining affected pieces. During the early game the average length of the list of affected pieces is 10-12, compared to an actual 25-30 pieces on the board. Thus, AFFLIST creates substantial savings in the time consumed doing position reevaluation.

To implement a move (to move forward on the search tree) the subroutine MOVE is used. The subroutine UNMOVE performs the inverse function to MOVE, therefore two are very useful in traversing the search tree.

B. CONTROL SECTION

The main program, T1, oversees the user-machine interface and provides certain control functions. It accesses several specialized subroutines to implement several of these control functions (see Figure 7). The subroutine DECODE performs decoding on the user move. The subroutine ENCODE provides algebraic notation print-
Figure 7. Control Section
out of moves stored in internal notation.

The subroutine LEGAL (checking move legality as its name suggests) assumes that MOVES contains all and only the current legal moves. Thus, LEGAL simply checks to see that the move under consideration is in MOVES.

The control section also provides certain convenience capabilities to the user. The ability to examine and modify IBD, MOVES, and/or IFLAG is provided. Additionally two printing subroutines, BORD and DRAW, are present. BORD draws the current position. DRAW draws the most recent search tree created by MARLA.

Tl also provides a mechanism which allows input of an arbitrary position for play (NPOS), a mechanism for printing the game history (MSTY), and a mechanism to facilitate user input of evaluation function control parameters.

C. MARLA

As discussed previously MARLA is the subroutine which controls selection of the machine move. Thus, MARLA is of particular interest. MARLA views the game as a concatenation of three subgames; the first an opening game, the second a middle game, and the third an end game.
The opening game \((16,18)\) is played by the subroutine \textsc{Opner} (see Figure 8), and involves tracing a tree of predefined openings, finding the recommended move, and making it. In a normal game \textsc{Marla} begins by consulting \textsc{Opner} for every move, until \textsc{Opner} fails or returns a last move indicator. \textsc{Opner} contains a tree frozen in a one-dimensional array. \textsc{Opner} uses a function, \(M\), to index into this array.

The endgame routine, \textsc{Endd}, determines which, if any, of the specific checkmating algorithms should be applied. \textsc{Endd} examines the board and returns unsuccessfully if there are more than ten pieces still in play. If there are fewer than 11, \textsc{Endd} packs these into a unique ten-digit number and examines that number for particular target values. If the board's number is not a target value then \textsc{Endd} returns unsuccessful, if it is then \textsc{Endd} calls the appropriate engame mating routine.

\textsc{Marla} evaluates positions using the evaluation function \textsc{Mefn}. \textsc{Mefn} combines four considerations using four user-modifiable weighting factors. The four considerations are: material, mobility, center control, and king safety.

If \textsc{Endd} returns unsuccessfully then \textsc{Marla} calls \textsc{Search}. \textsc{Search} is the general purpose move selector.
Figure 8. MARLA
and is the primary algorithmic part of MARLA (17,43). SEARCH implements a dynamic alpha-beta tree search with depth and width at user control.

The min-max (Shannon) tree search algorithm operates by moving the evaluation function values of terminal nodes up the search tree to the root. The value of any arc (move) in this search tree is set equal to the value of the node it enters (the son). By convention the value assigned to any white position (any position with white to move) is the maximum of the values of its immediate descendants. The white nodes will therefore be called maximizing nodes. Similarly, the black node values are the minima of their immediate descendants' values and the black nodes are called minimizing nodes.

The min-max algorithm can be augmented with the (30) alpha-beta heuristic. This heuristic terminates search below any minimizing node which has a min-max value less than any of its maximizing ancestors' values (a beta cutoff). Similarly, below any maximizing node with a value greater than one of its minimizing ancestor's value search may be terminated (an alpha cutoff). In this program these cutoffs occur during tree generation. Thus, a cutoff implies not merely that a section of the tree is not searched, but that that section is not even generated.
In order to determine which nodes to place in the tree, SEARCH calls SORT. SORT applies the evaluation function, MEPN, to the positions generated by each possible move and sorts the resulting move-value table by value. SEARCH then simply chooses the desired number of new tree nodes from the top of this list.
III. FEATURES AND STRATEGIES

Chapter III explores the context-free grammars which define this system's possible features and strategies. This chapter also develops the link between these grammars and our production system.

The production system is a pure character manipulation scheme. It involves no explicit flow-of-control information. It utilizes only pattern matching and string replacement capabilities. (Pattern matching in determining which productions to apply, and string replacement in applying that production.) Thus, the only manner in which a production system can do useful computation is one which involves attaching meaning to each production system symbol. In such a system the manipulation of symbols gains meaning through the individual symbols' meanings. Such a mechanism is therefore essentially semantic.

A general mechanism for formalizing this association of symbol and meaning is required in order to realize a useful production system. This mechanism already exists in formal grammars. Formal grammars (1) can be used to associate meanings with symbols. This potential is demonstrated by the grammars of Figures 9 and 12.
A. FEATURES

When a human chess player evaluates a position he will look for particular configurations of small manageable subsets of the set of all pieces on the board. He will seldom try to consider all pieces simultaneously. Each subset and its particular configuration are referred to as a feature of the chess position. Chess, having a rich and long heritage, has a large set of specific features known to every budding champion. One could easily develop algorithms to detect each.

However, such an independent, algorithmic approach would require the programmer to generate a recognizer for every possible feature. This approach would also require that the computer scientist define features; since the features would be individually programmed by him, and the program is, for the machine, the feature's definition. It would be preferable to allow the chess master to define the features directly to the machine, then let the machine generate the recognizer. All of the programs currently running contain algorithmically defined features.

It seems necessary from practical, theoretical, and modeling viewpoints that a computer chess program try to
recognize features at some stage. Theoretically the chess position can be viewed as a vector of 64 discrete variables. The theoretical problem (which is found in many areas) is that of associating with each state of the vector one of several possible actions or moves. There are too many vector states to consider a tabular association (the tree is too large). There is no closed form function which gives the proper association. Therefore, small, manageable subsets of the vector variable set are considered.

This model, rather than defining each feature independently, contains its own language for defining features in general. As mentioned, a feature can be thought of as a portion of the board with a certain piece configuration on it. Thus, features can be easily and straightforwardly described as sets of dependent conditions on the board values. Unfortunately, sets of dependent conditions are very hard to evaluate in general. Therefore, MARLA considers a feature to be the specifications defining a region and a set of independent conditions on that region. (A region is a set of board squares.) The two formulations are not equivalent, the former being more powerful, the latter being more useful (in a practical sense).

\(^3\)This language is a subset of English with terminals attached where necessary for parsing convenience.
MARLA views a region in a partially passive rather than in a completely constructive sense. A region to MARLA is a filter over which each of the 64 squares of the board is passed, and a set of increments to be added to the row and column indices of the squares which drop through.

A powerful mechanism for defining chess features in this context is the formal grammar. The grammar used by this program to define chess features is given in Figure 2. This grammar associates with each feature definition a symbol to be used by the production system and a recognizer. This grammar is a context-free recursive translation grammar (1).

Parsing arbitrary context-free grammars can be difficult. This problem can be avoided by using the technique of 'terminal saturation'. In this technique numbered terminals are inserted around all non-terminals on production right-hand sides. By demanding that all productions with the same left-hand sides begin their right-hand sides with unique terminals it is easy to make the grammar parsable in linear time (1).

The recognition of feature descriptions and the association of each with a symbol are not sufficient. The machine must be able to recognize the occurrence of a feature in actual positions. To do so MARLA requires
Feature ::= (1) Keysquare (2) Ralist (3)
Ralist ::= (4) Region (5) Attributes (6)
Ralist ::= (7) Region (8) Attributes (9) Ralist (10)
Region ::= (11) Increments (12)
Region ::= (13)
Region ::= (14) Dinc (15) Increments (16)
Increments ::= (17) Rinc (18) Cinc (19) Increments (20)
Increments ::= (21)
Rinc ::= (22) Number_{n,s} (23) n=1
Rinc ::= (24) Number_{n,s} (25) n=2
Rinc ::= (26)
Cinc ::= (27) Number_{n,s} (28) n=3
Cinc ::= (29) Number_{n,s} (30) n=4
Cinc ::= (31)
Dinc ::= (32) / (33) / (34)
Keysquare ::= (35) Rspec (36) Cspec (37) Vspec (38) Dspec (39)
Dspec ::= (40) / (41) / (42)
Vspec ::= (43) Val_{i,q} (44) i=5

Figure 9. Grammar defining feature description language.
\[ V_{\text{spec}} ::= (45) \text{Conspec} (46) \]
\[ \text{Conspec} ::= (47) \text{Val}_{i,q} (48) \]
\[ i=6 \]
\[ V_{\text{spec}} ::= (50) \]
\[ \text{Conspec} ::= (49) \]
\[ R_{\text{spec}} ::= (51) \text{Val}_{i,q} (52) \]
\[ i=7 \]
\[ R_{\text{spec}} ::= (53) \]
\[ \text{Val}_{n_1,q} ::= (54) \text{Number}_{n,s} (55) \]
\[ q=s \]
\[ \text{Val}_{n_1,q} ::= (56) \text{Number}_{n,s} (57) \text{Specialchar}_{k} (58) \]
\[ q=s^k \]
\[ \text{Specialchar}_{k} ::= (59) / (60) \]
\[ k=15 \]
\[ k=-15 \]
\[ \text{Conspec} ::= (61) \text{Val}_{i,q} (62) \text{Conspec} (63) \]
\[ i=5 \]
\[ \text{Conspec} ::= (64) \text{Val}_{i,q} (65) \]
\[ i=5 \]
\[ \text{Attributes} ::= (66) \text{Spec} (67) \text{Opp} (68) \text{Val}_{i,q} (69) \]
\[ i=5 \]
\[ \text{Attributes} ::= (70) \text{Spec} (71) \text{Opp} (72) \text{Val}_{i,q} (73) \]
\[ i=5 \]
\[ \text{Attributes} (74) \]
\[ \text{Opp} ::= (75) \text{Oper}_{j} (76) \]
\[ j=1 \]
\[ \text{Opp} ::= (77) \text{Oper}_{j} (78) \]
\[ j=2 \]

Figure 9. Continued
Oper \(_j\) ::= (79) Op\(_{p,m}\) (80) 
    p=1
    m=j

Oper \(_j\) ::= (81) Op\(_{p,m}\) (82) 
    p=2
    m=j

Op\(_{p,m}\) ::= (83) / (84) / (85) / (86) / (87)

Spec ::= (88) Var (89) Number\(_n,s\) (90) 
    n=3

Var ::= (91) / (92) / (93)

Number\(_n,s\) ::= (94) / (95) / (96) / (97) 
    s=1 \hspace{1cm} s=2 \hspace{1cm} s=3 \hspace{1cm} s=4

Number\(_n,s\) ::= (98) / (99) / (100) / (101) 
    s=5 \hspace{1cm} s=6 \hspace{1cm} s=7 \hspace{1cm} s=8

Note: Number\(_n,s\) involves many more productions whose 
form is identical to those above.
feature recognizers as well as a feature description recognizer. The features are defined while running MARLA in a training mode (see Figure 1), thus the recognizers can be produced no sooner than this. The recognizers are produced by action symbols associated with each terminal in the feature grammar. The grammar therefore is formally a translation grammar. It accepts feature definitions and produces feature recognizer tables. Since the recognizers are table-driven the action symbols need only store information in these tables during the feature definition's parse. The action strings associated with each terminal are given in Table 10.

In Figure 9 there are also attributes associated with certain non-terminals in the grammar. These are of two types: synthesized and inherited. A synthesized attribute is used to bring information obtained at a leaf up the tree to an ancestor node to be stored by that node in the recognizer table. These are required when information from several places must be combined before storage.

Inherited attributes on the other hand, carry information down the tree to aid in parsing at a descendant node. For example, \( \text{Number}_{n,s} \) can be called from several places. If called from Rspec, \( \text{Number}_{n,s} \) must look for a
<table>
<thead>
<tr>
<th>terminal number</th>
<th>related attribute value</th>
<th>English translation</th>
<th>action symbol string</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>A SQUARE</td>
<td>Create KeySquare filter pointer</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>Create feature list entry</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>Put region in increments table</td>
</tr>
<tr>
<td>4</td>
<td>AND A REGION CONSISTING OF</td>
<td></td>
<td>novar = 0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>Link attribute table</td>
</tr>
<tr>
<td>6</td>
<td>AND A REGION CONSISTING OF</td>
<td></td>
<td>Put region in increments table</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>novar = 0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>Link attribute table</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>Flag = 0</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>tvop = 0</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>ltop = 1</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>THE SQUARES THE PIECE THERE CAN MOVE TO</td>
<td>flag = 1</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>A SQUARE</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>THE SQUARE(S)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>AND</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>ttop = ttop+1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>ltop = ltop</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>ttop = ttop-1</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td>T(l, ttop) = s</td>
</tr>
</tbody>
</table>

Table 10, Terminal symbol translation table
\[ T_{(1,t_{\text{top}})} = T_{\text{fun}(s)} \]
\[ I = -8, 8: \]
\[ T_{(1,t_{\text{top}})} = 1 \]
\[ t_{\text{top}} = t_{\text{top}} + 1 \]
\[ I = l_{\text{top}}, t_{\text{top}}: \]
\[ T_{(2, I)} = s \]

\[ I = l_{\text{top}}, t_{\text{top}}: \]
\[ T_{(2, I)} = T_{\text{fun}(s)} \]
\[ I = -8, 8: \]
\[ T_{(2, l_{\text{top}})} = 1 \]
\[ T_{(1, l_{\text{top}})} = T_{(1, t_{\text{top}})} \]
\[ l_{\text{top}} = l_{\text{top}} + 1 \]
\[ \text{dval} = \text{dval} + 1 \]
\[ \text{dval} = \text{dval} + 2 \]
\[ I = 1, 3: \]
\[ \text{valvei} = 0 \]

\[ \text{dval} = 1 \]
\[ \text{dval} = 2 \]
\[ \text{dval} = 3 \]

\[ \text{valve3} = 1 \]
\[ \text{TAB}(3, 1) = q \]

\[ \text{valve2} = 1 \]
\[ \text{TAB}(2, 1) = q \]

\[ \text{valve1} = 1 \]
\[ \text{TAB}(1, 1) = q \]

\textbf{Table 10} . Continued
ON IT OR
valve3 = valve3 + 1
TAB(3, valve3) = q

ON IT
valve3 = valve3 + 1
TAB(3, valve3) = q

SUCH THAT
novar = novar + 1

SUCH THAT
ATAB(4, novar) = q

AND
ATAB(4, novar) = q

IS
ATAB(5, novar) = 0

IS NOT
ATAB(5, novar) = 1

ATAB(6, novar) = 0

ATAB(6, novar) = 1

ON
ATAB(3, novar) = 1
ON
ATAB(3, novar) = 1

THE SAME PIECE AS
ATAB(3, novar) = 1
TO THE RIGHT OF
ATAB(3, novar) = 2
TO THE QUEEN'S SIDE OF
ATAB(3, novar) = 2
STRONGER THAN
ATAB(3, novar) = 2
RELATIVELY STRONGER THAN
ATAB(3, novar) = 2

ON OR TO THE LEFT OF
ATAB(3, novar) = 5
ON OR TO THE QUEEN'S SIDE OF
ATAB(3, novar) = 5

Table 10. Continued
### Table 10. Continued

<table>
<thead>
<tr>
<th>Number</th>
<th>Description</th>
<th>ATAB(3, novar)</th>
</tr>
</thead>
<tbody>
<tr>
<td>87</td>
<td>2.1 Equal to or weaker than</td>
<td>ATAB(3, novar) = 5</td>
</tr>
<tr>
<td>88</td>
<td>2.2 The same or relatively weak than</td>
<td>ATAB(3, novar) = 5</td>
</tr>
<tr>
<td>89</td>
<td></td>
<td>ATAB(1, novar) = s</td>
</tr>
<tr>
<td>90</td>
<td>The row of man</td>
<td>ATAB(2, novar) = 1</td>
</tr>
<tr>
<td>91</td>
<td>The column of man</td>
<td>ATAB(2, novar) = 2</td>
</tr>
<tr>
<td>92</td>
<td>Man</td>
<td>ATAB(2, novar) = 3</td>
</tr>
<tr>
<td>93</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>One row: towards white</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>One row ahead</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>One column right</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>One column to the king's side of the king's side of</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>Queen's rook's</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>First</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>One</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>In the same row</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>In the same row</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>In the same column</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>In the same column</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>Same</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>Eighth</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>Eighth</td>
<td></td>
</tr>
</tbody>
</table>

...
row specification, whereas, when called from Vspec, Number must look for a piece specification. (Hence the values associated with a terminal may depend on the value of an attribute inherited from the non-terminal which generated it.) By keeping the attribute usage simple their evaluation is kept simple.

Consider now the example begun in Chapter I. The feature 'self-controlled center' is described in the machine's current input language as:

\[(1) (35) (51) (54) (97) (55) (52) (36) (47) (54) (97) (55) (48) (37) (49) (38) (41) (39) \ldots (3)\]

The first portion of the parse tree for this feature is given in Figure 11. This figure demonstrates the attribute evaluation process as well. The key square filter passes any square whose row and column indices are both four. The region is then defined as the typical center (squares \((4,4),(4,5),(5,4),\) and \((5,5)\)).

It should be clear that parsing is trivial using this grammar. During parsing, attributes are passed down

---

5 If one were attempting a natural language parse then the former would imply looking for 'FIRST RANK', 'SECOND RANK', etc.. The latter would imply looking for 'KING', 'QUEEN', 'PAWN', etc..

6 The more readable English language version is:

A SQUARE ON THE FOURTH RANK IN THE FOURTH COLUMN
Figure 11. Partial parse tree for feature: self-controlled center
the tree during descent and back up the tree during ascent. This system is designed in such a manner that the two attribute types never interact in a manner which inhibits the parsing process.

During the recognition process an action string is produced. This string has three sections. The first builds the key square filter. The second produces increments lists, (one in this case) and the third creates an attributes table. During play these three sections act as a feature recognizer.

As a final note on features, the feature recognizers produced by the machine are themselves independent, and therefore their run time consumption is linear in their number.

B. STRATEGY IN MARLA'S PERSPECTIVE

Theoretically, the recognition of a feature and the implementation of a strategy are very similar. In the latter case one can examine a difference vector specifying a move's effect. In the former one examines a total state vector representing a position. With this view implementation of a strategy involves determining which moves' difference vectors satisfy a set of strategy conditions, just as recognizing a feature involves determining which
subsets of the total state vector satisfy a set of feature conditions.

In MARLA's view the difference vector contains only a move representation and the list of new moves available to the moved piece. This vector is considerably smaller than the total state vector involved in feature recognition. The grammar developed to represent strategies exhibits this simplicity (see Figure 12).

After examining Figure 12 the reader should realize that this grammar defines a finite language. Therefore, the possibility of implementing all strategies with a single routine exists, and such a routine has been incorporated into this program.

The grammar of Figure 12 again demonstrates the terminal saturation technique. All attributes are synthetic. The trees this grammar fosters all have the property that complete information about any parse is drawn up the tree and finally encoded in the synthetic attribute of the starting non-terminal. Note: the

---

9 Consider the benefits of these terminal saturated grammars:

1. The grammar is easily parsable, even LL(1).
2. The terminal implementation; parser symbols, English language equivalents, and action symbols, can all be consolidated into one uniform routine.
3. Implementation of a top down parser is simple (mechanistic) and uniform using the routine of benefit 2 above and the actual grammar description.
Strategy := (1) Action, THE Area, WITH Modifier \( m \) (4)

\[ s \in \text{S} \]

**Action** := CAPTURE
\( a \in \text{CA} \)

**Action** := ATTACK
\( a \in \text{AT} \)

**Action** := MOVE
\( a \in \text{MV} \)

**Action** := MOVE INTO
\( a \in \text{MI} \)

**Action** := ADVANCE
\( a \in \text{AD} \)

**Action** := RETREAT
\( a \in \text{RE} \)

**Action** := DEFEND
\( a \in \text{DE} \)

**Area** := MOV SQUARE
\( c \in \text{S} \)

**Area** := AREA
\( c \in \text{R} \)

**Area** := PIECE
\( c \in \text{I} \)

**Modifier** := (15)
\( m \in \{500\} \)

**Modifier** := PAWN
\( m \in \{100\} \)

**Modifier** := PIECE
\( m \in \{500\} \)

**Figure 12.** Strategy defining grammar
$\text{Modifier}_{n} ::= \text{FORCES Attribute}_{1} \ (19)$

\[ n=1 \]

$\text{Attribute}_{a} ::= \text{OF Pcode}_{n} \text{ STRENGTH}$

\[ a=1n \]

$\text{Attribute}_{a} ::= \text{OF GREATER THAN Pcode}_{n} \text{ STRENGTH}$

\[ a=2n \]

$\text{Attribute}_{a} ::= \text{OF LESS THAN Pcode}_{n} \text{ STRENGTH}$

\[ a=3n \]

$\text{Pcode}_{n} ::= \text{QUEEN}$

\[ n=1 \]

$\text{Pcode}_{n} ::= \text{ROOK}$

\[ n=R \]

$\text{Pcode}_{n} ::= \text{BISHOP}$

\[ n=B \]

$\text{Pcode}_{n} ::= \text{KNIGHT}$

\[ n=N \]

$\text{Pcode}_{n} ::= \text{PAWN}$

\[ n=P \]

$\text{Pcode}_{n} ::= \text{KING}$

\[ n=K \]

\[ n=3 \]

\[ n=4 \]

\[ n=5 \]

\[ n=6 \]

\[ n=7 \]

\[ n=8 \]

\[ n=9 \]

\[ n=10 \]

\[ n=11 \]

\[ n=12 \]

\[ n=13 \]

\[ n=14 \]

\[ n=15 \]

\[ n=16 \]

\[ n=17 \]

\[ n=18 \]

\[ n=19 \]

\[ n=20 \]
attributes are all string variables in this grammar, and
the implied operation is concatenation, not multiplication.
To explain these points, consider the strategy of our
example maxim:

you control the center
IF and you have a mobile THEN that wing
pawn wing with pawns

The strategy involved is given to MARLA as (see Figure 12):

(1) (9) (2) (13) (3) (16) (4)

As the input string is parsed the tree of Figure 13 is pro-
duced. During parsing the value AD-R-POO is synthesized
for attribute 's' of Strategy . This value will be used
in the production system as a token representing this
strategy.

This value is also used by the implementation routine.
All information about the strategy needed by the imple-
mentation routine is given in the token itself. In order
to implement the strategy during a game the implementation
routine must also have game-time information on actual
region values. This additional information is passed to
the strategy implementation routine through the production
system by the feature recognizer which recognized it.

In this particular case at game-time the mobile
pawn wing recognizer will pass to the strategy implementa-
Figure 13. Parse tree for example strategy
routine the actual squares on the open left wing (see Figure 2).
IV. THE PRODUCTION SYSTEM

The master chess player will consider a deep tree yet one with few nodes. To do so he must have a powerful mechanism for sorting the possible continuations at any node and choosing the proper few to put in the tree. A computer chess program generally searches a very large tree. However, the computer's tree is generally more shallow than the master's and the computer's level of play is uniformly far below the master's. Increasing the computer's search speed alone will not overcome the exponentially increasing move population. Nonetheless, efforts have been and will be made to apply brute force to the chess tree. These efforts, often termed "technology programs" include notably MANIAC (19) and TECH (13) and serve as benchmarks by which to measure other chess programs and technological advance.

Rather than faster tree searching techniques, better move sorting and selection techniques are necessary. Such reasoning has inspired the work of A. L. Zobrist and F. R. Carlson. When master chess players are asked how they sort so effectively they, after reminding you that the question has no simple answer, will begin quoting chess maxims. The intent here is to sort the moves available to the tree searcher by applying these chess maxims formally. The
mechanism used to accomplish this is a production system. Each maxim will be captured in a production or several productions of the system. Since the number of individual chess maxims a chess master may use in a game is very large, the eventual number of productions in this system will be very large.

A. PRODUCTION SYSTEM OPERATION

Post's original, normal form production (29) contains only an implied string replacement. The form is:

\[ gP \rightarrow Pg' \]

meaning string \( Pg' \) replaces string \( gP \). For the present purposes this has been changed to the form:

\[ (F:AB...KXY...Z) \rightarrow W \]

This can be interpreted as:

\[ \text{IF} (F:AB...KXY...Z) \text{ THEN } W \]

or: \( \text{IF} (\text{features}) \text{ THEN } (\text{strategies}) \)

where: \( W = \text{strategies} \)
\( A...K = \text{features} \)
\( X...Z = \text{strategies} \).

These productions replace exactly one element of the left-hand side, the first, with a set of strategy tokens. A production is applicable iff all elements of its left-hand
side appear in the current string irrespective of their order of appearance. Thus, to implement the above production the current string is examined for an occurrence of each of the features A, B, ... F, ... K and an occurrence of each of the strategy taken X, ... , Z. If all are present then the feature F is removed from the string and the strategy W is added to the string. Application of any particular production will alter the current string in such a manner that the production itself will not cause the production to enter an infinite loop.

As indicated, each production can be viewed as an IF (features) THEN (strategies) statement. The THEN clause contains a string of strategy tokens each of which is to be placed in the current string.

In a previous chapter a mechanism for defining features was developed. If during analysis of a position an instance of a feature is found then the symbol representing that feature is placed in the production system's initial string. The production system then operates cyclically replacing features with strategies until it finds no more maxims to be useful (productions to be applicable). At this point the production system output string is a list of features and strategies. (Strategies were considered in greater detail in Chapter III.) Having thus chosen a set of strategies MARLA can use these
strategies to sort the moves available. MARLA then builds
the search tree selecting nodes at each level on the basis
of such sorts.

B. LINES OF PLAY IN CHESS

A chess master when explaining how he develops his
small but accurate tree (21) will also speak of lines and
line refutations. In chess a line is a series of moves
all with similar intent. With the strategy oriented
sorting mechanism described here, lines themselves should
be realizable.

This move to move uniformity of strategy appears in
the tree as uniformity between alternate levels. In order
to achieve this uniformity MARLA forces certain strategies
which appear at one level of the search tree to appear
also at the alternately next level. To facilitate this
MARLA must record for each node which strategies recom-
mended that node's immediate descendants.

There are many types of chess maxims. Using this
schema any maxim which can be phrased as an if-then
statement can be represented. Examples are:

IF (YOUR QUEEN IS JEOPARIZED) THEN (MOVE THE PIECE)

and: IF (YOUR OPPONENTS PIECE IS PINNED)
THEN (ATTACK THE PIECE WITH FORCES OF PAWN STRENGTH).
A more complex example which fits this mold is:

\[
\text{IF } ((\text{THE CENTER IS OPEN}) \text{ AND } (\text{ATTACK THE AREA WITH FORCES OF PAWN STRENGTH}))
\]

\[
\text{THEN DON'T}^9 \text{(MOVE INTO THE AREA WITH FORCES OF GREATER THEN PAWN STRENGTH).}
\]

C. OPERATIONAL ENHANCEMENTS

There is another area which must be considered. It involves the running of the production system itself. Two questions must be addressed: one, how are multiple production loops and production conflicts resolved; and two, how are the resulting strategies combined.

Consider the latter question. When several

\[^9\text{I have not yet mentioned negating a feature or a strategy. A strategy is a routine which when given a non-zero weight by the production system selectively distributes that weight among a set of moves such that the moves which receive the largest portions best implement that strategy. A negated strategy (i.e. DON'T strategy) is simply a strategy given a negative weight. A feature on other hand, is a routine which for any given position will return a true (the feature is present) or a false (the feature is not present). Thus, in order to negate a feature one must complement the value its recognizer returns. In order to recognize a negated feature in the current production system string one checks for its presence and complements the results. A negated strategy however, must have a negative weight and be present in the current string. MARLA's instructor must bear this in mind when using the phrases: IT IS NOT THE CASE THAT and DON'T. The former negates a feature, and the latter negates a strategy.}\]
strategies are simultaneously active some mechanism must be present to direct their interaction. Attempts have been made to incorporate the manipulation of weighting factors (measures of belief; see especially 37) in production systems. In these systems the weights are usually generated by the user and manipulated only in predefined ways which are thought to simulate human judgement. In this system weighting factors are used, however, they are developed automatically.

Each strategy token in the production system current string is associated with a weight factor. Appended to each production in the system is an action string. That is, we have the form:

\[(P:AB...KXY...Z) \rightarrow W(\text{action string})\]

where \((\text{action string}) = \text{mainweight (strategy,}I)\). The action string is interpreted as meaning: "if the first strategy of the right-hand side is not in the current string then use 'mainweight' as the initial weight for that strategy else, add mainweight to that strategy's current weight". For each strategy token in the action string; if the strategy is in the current string then add 1 to its current value else, put it in the current string with 1 as its initial weight. At the termination of the production system node evaluation each strategy suggested will have
an associated weight. Each strategy then distributes its weight among the moves available at that node.

Several numbers thus gain significance. The total weight apportioned during move analysis at a node will be referred to as the operational weight total. The minimum acceptable move weight (the minimum weight which will get a move into the tree) will be referred to as the cutoff weight (or just CW). Any strategy which has been given a weight greater than or equal to CW (i.e. a weight great enough to force one move to appear in the tree) is called an R strategy. Any production which can give a particular strategy an initial weight or a weight increment greater than CW is called an R production.

Chess maxims come in many degrees of significance. Some state absolute imperatives, some suggest nuances. At the time each maxim is suggested to MARLA an indication is given of the strategy's degree of significance. MARLA uses this to place the maxim in one of three distinct classes which partition the set of all maxims. These three are arbitrarily labeled the S maxims, the H maxims, and the R maxims. The R maxims must generate R productions. The H maxims must be given greater main weights than the S maxims, and the H maxims precede the S maxims in order of
application. R maxims are applied before H maxims. The initial approximations to the production main weights may come from the user. MARLA's job is to refine these initial weights.

Before turning to the topic of MARLA's study of master games, return for a moment to the problem of multiple production loops. MARLA limits the number of times any production can fire during analysis at one node. The production system is therefore guaranteed (by finiteness of the production set) to complete the analysis of any position. Generally one analyzes production system behavior in terms of the list of productions applied in their order of application. MARLA analyzes these lists to detect loops. MARLA also analyzes unordered subsets of the productions applied, one subset associated with each strategy in the final production system string. When during production system operation MARLA must interrupt to enforce the number of applications limit, it places a loop mark in the production list and inhibits that production for the rest of the current node's evaluation. Furthermore, during operation each production which affects the weight of a strategy must leave a signature and a copy of its effect in a strategy production signature list. Thus each strategy has a record of the productions which recommended it and their
recommendations.

D. TRAINING MODE

MARLA has a training mode. In this mode MARLA examines the lists of productions associated with each non-terminal node. The area preceding each loop mark left during production system execution is examined for loops. In this training mode MARLA can also follow master games and correct her production system weights using these games.

When the strategy implementation routine evaluates the moves possible at a node each move receives weight from a set of strategies. Just as the productions must sign in when they give weight to strategies, so strategies must sign in when they give weight to moves. The entire situation is diagrammed in Figure 8. MARLA can therefore examine a position, analyze it, and select a move. Having selected a move the program can check it against the master's move in that position. If they agree, the move is made. If not the master's move is made and the production signature and strategy signature tables are examined. The master's move and the strategies which recommended it (if any) are found. Then the set of productions responsible for those strategies is generated. Having found the relevant
Production System Cycle

get production

apply production

sign production
signature table

PS done?

yes

get active strategy

apply strategy

signature table

done

weight increment

feature and/or strategy tokens

current strategy weights

PRODUCTION SIGNATURE TABLE

production signatures

pointers to crux regions

strategies

weight increment

moves list

STRATEGY SIGNATURE TABLE

Figure 14. Learning mechanism structures
productions MARLA can alter their mainweights and/or increments in such a manner that the master's move would have been chosen, or at least would have received more weight. During this process MARLA has an accepted move (the master's) for only her first search ply (one tree level), therefore when in training mode MARLA searches only one ply deep.

Thus, MARLA rewards its production system for placing the master's move in its tree. MARLA can modify the production system coefficients by direct examination of games. The only necessary requirement of this method is that MARLA's modifications be convergent.

E. CROSS PRODUCTION SYSTEM COMMUNICATION

Now consider the question of communication between the feature recognizers and the strategy implementation routine. Included in the description of each feature is, as well as the specifications mentioned previously, an indication of which region (or key square) is to be passed to a strategy. This region is called the crux or crux region. When a position is analyzed by the feature recognizers, a token is put in the production system initial string for each instance recognized. Associated
with each token is a pointer to the crux of the token's instance. During production system execution this pointer is associated with the strategy token which replaces its feature token in the current string. Thus at production system termination each strategy has a list of crux regions associated with it. The list is used when evaluating conditions relevant to the strategy. Each strategy-crux pair has a particular weight associated with it inherited from the production system which created the pair.

Finally, return to the question of conflict resolution. When two or more productions are simultaneously applicable to the current string, these productions are said to be in conflict. In this system there are four classes of productions. In conflict resolution priority order these are the system productions, the R productions, the H productions, and the S productions. For simplicity conflicts within each class are resolved alphabetically.
V. CONCLUSION

Practical machine representation of knowledge must use both algorithmic and semantic approaches. That algorithmic implementations alone are insufficient is an hypothesis greatly supported by the abyssmal performance of current computer chess programs. That semantic representations alone are insufficient is not so clear, particularly given our human semantic data base. Nonetheless, considering the primitive state of semantic understanding by computers; purely semantic methods are inadequate. This latter inadequacy is due in part to the intrinsically algorithmic structure which exists in problems and in part to the slow, bulky condition of our present semantically oriented systems.

In support of the thesis that the two approaches can be used together effectively a system has been developed which uses both. The interfacing mechanism between the system sections embodying each approach is the attributed translation grammar. The system developed here has been applied to the game of chess, thereby generating a program which not only implements chess but also serves as a production system model of the human thought processes involved in chess learning and analysis.
The model not only exhibits learning behavior in the form of advice-taking, but also has primitive mechanisms for modeling the mental refinement of advice taken and the idea of mental theme.

The intent is not and has not been that of presenting an unbeatable chess program. Many well prepared groups have spent much time trying, unsuccessfully, to do so. Rather, this thesis presents a program which plays chess, and which can learn to play better and better.
BIBLIOGRAPHY


17. Horowitz, Mott-Smith,Point Count Chess, David McKay Inc.,N.Y.,1960


37. Shortliffe, E.H., (1976), Computer Based Medical Consultation: II: CII, American Elsevier,


43. Znosko-Borovsky, E.A., (1960), How NOT to Play Chess, Dover, N.Y.
