

Using Plans in Chess

David Wilkins
Stanford Artificial Intelligence Laboratory
Stanford University
Stanford, California

The purpose of this research is to investigate the extent to which knowledge can replace and support search in selecting a chess move and to delineate the issues involved. This has been carried out by constructing a program, PARADISE (PAttern Recognition Applied to DIrecting SEarch), which finds the best move in tactically sharp middle game positions. It encodes a large body of knowledge in the form of production rules. The program uses the knowledge base to discover plans during static analysis and to guide a small (tens of nodes) tree search. PARADISE does not place a depth limit on the search (or any other artificial effort limit), and has found combinations as deep as 19 ply. This paper describes plans, how they are produced, and how they are used to guide the search. A complete description of PARADISE appears in [12].

1. Introduction

One of the central concerns of artificial intelligence is expressing knowledge and reasoning with it. Chess has been one of the most popular domains for AI research, yet brute force searching programs with little chess knowledge play better chess than programs with more chess knowledge. There is still much to be learned about expressing and using the kind of knowledge involved in playing chess. CHESS 4.7 is probably the best chess program, and its authors make the following comments in [11]:

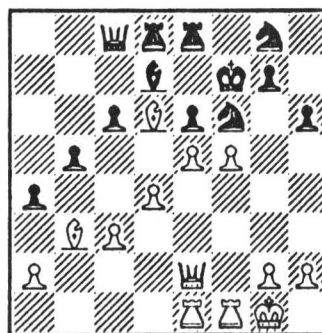
But why is the program so devoid of chess knowledge? . .

Our problem is that the programming tools we are presently using are not adequate to the task. Too much work is needed to encode the sizable body of knowledge needed to significantly improve the quality of play.

Human masters, whose play is still much better than the best programs, appear to use a knowledge intensive approach to chess (see [2]). The purpose of this research is to investigate the issues involved in expressing and using pattern-oriented knowledge to analyze a chess position, to provide direction for the search, and to communicate useful results from the search. To reduce the amount of knowledge that must be encapsulated, the domain of the program has been limited to tactically sharp middle game positions. The phrase "tactically sharp" is meant to imply that success can be judged by the gain of a material rather than a positional advantage. The complexity of the middle game requires extensive search, providing a good testing ground for attempts to replace search with knowledge and to use knowledge to guide the search and communicate discoveries from one part of the tree to another.

Since, in our chosen domain, the attacking side can generally win material with correct play, the program uses different models for the offensive and defensive players. The offensive player uses a large amount of knowledge, suggests many esoteric attacks, and invests much effort in making sure a plan is likely to work before suggesting it. The defensive player uses less knowledge and effort, tries any defense that might work, and tries only obvious counter attacks. Trying every reasonable defense enables the offense to satisfy itself that a line really does win. The size of the tree is kept small because the offensive player often finds the best move immediately.

For a knowledge based program to achieve master level performance, it seems essential that the knowledge base be amenable to modifications and additions. PARADISE provides for easy modification of and addition to its knowledge base (without adversely affecting program performance), by having the knowledge base composed of largely independent production rules which are written in a straightforward Production-Language (see [12]). The goal is to build an expert knowledge base and to reason with it to discover plans and verify them with a small tree search. As long as this goal is reached, the amount of execution time used is not important. When a desired level of performance is reached, the program could be speeded up considerably by switching to a more efficient representation at the cost of transparency and ease of modification.



1. Q-R5ch NxQ
2. PxPch K-N3
3. B-B2ch K-N4
4. R-B5ch K-N3
5. R-B6ch K-N4
6. R-N6ch K-R5
7. R-K4ch N-B5
8. RxNch K-R4
9. P-N3 any
10. R-R4 mate

Figure 0 - white to move

Because it grows small trees, PARADISE can find deeper combinations than most chess programs. Figure 0 is an example of such a combination from an actual master game. White can mate by sacrificing his queen and playing nine additional moves (against black's line of longest resistance), the next to last move not being a check. Thus the search tree must go at least 19 ply deep to find this combination. This is considerably deeper than any other current chess program could probe in a reasonable amount of time, but PARADISE solves this problem after generating a search tree of 109 nodes, in 20 minutes of CPU time on a DEC KL-10.

2. Overview of PARADISE

PARADISE uses a knowledge base consisting of about 200 production rules to find the best move in chess positions. Every production has a pattern (i.e., a complex, interrelated set of features) as its condition. Each production can be viewed as searching for all instances of its condition pattern in the position. For each instance found, the production may, as its action, post concepts (described in the next section) in the data base for use by the system in its reasoning processes. The data base can be considered as a global blackboard where productions write information. The productions are built up by levels, with more primitive patterns (e.g., a pattern which matches legal moves) being used as building blocks for more complex patterns. A search tree (the major component of most chess programs) is used to show that one move suggested by the pattern-based analysis is in fact the best.

To offset the expense of searching for patterns in the position, problems must be solved with small search trees. In fact, PARADISE can be viewed as shifting some of its search from the usual chess game tree to the problem of matching patterns. PARADISE shows that its pattern-oriented knowledge base can be used in a variety of ways to significantly reduce the part of the game tree which needs to be searched. This reduction is large enough that a high level of expertise can be obtained without placing a depth limit (or any other artificial effort limit) on the search. The various ways in which the knowledge base is used are briefly mentioned below.

Calculating primitives. PARADISE's knowledge base contains about twenty patterns (productions without actions) which are primitives used to describe a chess position. These patterns are matched once for each position. They range from matching legal moves to matching any move which can be made without immediate loss of material. The primitives are relatively complex and calculating them is expensive.

Static analysis. Given a new position, PARADISE does an expensive static analysis which uses, on the average, 12 seconds of cpu time. This analysis uses a large number of productions in the knowledge base to look for threats which may win material. It produces plans which should guide the search for several ply and should give information which can be used to stop the search at reasonable points.

Producing plans. The knowledge base is used to produce plans during static analysis and at other times. Through plans, PARADISE uses its knowledge to understand a new position created during the search on the basis of positions previously analyzed along that line of play, thus avoiding a static analysis. Plans guide the search and a static analysis is only occasionally necessary in positions created during the search. Producing a plan is not as simple as suggesting a move, it involves generating a large amount of useful information to help avoid static analyses along many different lines and to help detect success or failure of the plan as it is executed. (Examples are given later.)

Executing plans. The execution of a plan during the tree search requires using the knowledge base to solve goals specified in the plan. This use of the knowledge base could be viewed as a static analysis where the system's attention is focused on a particular goal or aspect of the position. Such a focused analysis can be done at a fraction of the cost of a full analysis which examines all aspects of the position.

Generating defensive moves. PARADISE does static analyses and executes plans only for the offense (the side for which it is trying to win material). To prove that an offensive plan succeeds, the search must show a winning line for all reasonable (i.e., any move which might be effective) defensive moves. Productions in the knowledge base are used to generate all the reasonable defenses.

Quiescence searches. When a node is a candidate for termination of the search, PARADISE uses a quiescence search to determine its value. This quiescence search may return information which will cause the termination decision to be revoked. Productions in the knowledge base provide much of the knowledge used in PARADISE's quiescence search. Quiescence searches in PARADISE investigate not only captures but forks, pins, multi-move mating sequences and other threats. Only one move is investigated at each node except when a defensive move fails.

Analysis of problem upon failure. When either the defense or offense has tried a move that failed, the search analyzes the information backed up with the refutation in an attempt to solve the problem. The knowledge base is used in this analysis to suggest plans which thwart the opponent's refutation. In this way PARADISE constructs new plans using the information gained from searching an unsuccessful plan.

Answering questions. The search often requests certain kinds of information about the current position. The knowledge base is used to provide such information. The most frequent requests ask if the position is quiescent and if there are any obviously winning moves in the position.

The above list describes eight general functions the knowledge base performs for PARADISE. This paper explains how plans are produced and executed. The productions, the static analysis, the tree search, and mechanisms for communicating information from one part of the tree to another are discussed in [12].

3. The Need for Concepts

To understand a chess position, a successful line of play must be found. Human masters understand concepts which cover many ply in order to find the best move. A knowledge based program must also have the ability to reason with concepts that are higher level than the legal moves in a chess position. The following position from [1] illustrates this.

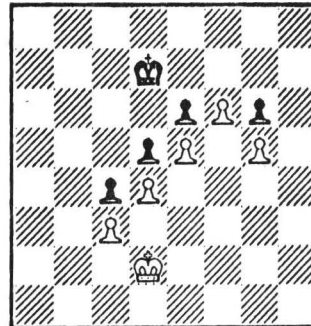


Figure 1 - white to move

In the above position, most computer chess programs would search the game tree as deep as their effort limit allowed and decide to play K-K3 which centralizes the king. They cannot search deep enough to discover that white can win since white's advantage would take more than 20 ply to show up in most evaluation functions used by computer programs. The point is that the solver of the above problem uses conceptual reasoning to solve it, and the concepts must be higher level than the legal moves in a chess position. A human master would recognize the following features (among others) in the above position: the black king must stay near his KB1 to prevent white's king bishop pawn from queening, the pawns form a blockade which the kings can't penetrate, and the kings can bypass this blockade only on the queen rook file.

For a knowledge based program to solve this problem, it must be able to express concepts at the level of conceptualization used above. The system must combine and reason with these concepts in unforeseen ways. In the above example, the blockading concept would be used to develop the plan of moving the white king to the queen rook file in order to bypass the blockade, but if both black and white had pawns on their QN4 then the blockading concept would be used to decide that neither king can invade the other's territory. Expressing and using concepts is a major problem for a knowledge based chess program.

To clarify the problems involved in using concepts to reason about chess, a comparison to a well-known "expert" system such as MYCIN is helpful. The following phrase is the action part of a typical MYCIN rule (from [3]): *then there is suggestive evidence (.7) that the identity of the organism is bacteroides*. Such an action effectively adds to the plausibility score for some possible answer. This presupposes that any one writer of all the rules knows every possible solution since such rules will never create a diagnosis that has not been mentioned in the action part of some rule. Such an approach is not satisfactory for the type of reasoning needed in figure 1.

Using actions similar to those used in MYCIN rules would be similar to a chess system where the action part of each rule was to add to the plausibility score of either 1) one or more of the legal moves in the position, or 2) some prespecified object which could lead (through other rules) to recommending a legal move. Such a system would not be able to reason that a king must get past a pawn blockade to attack an opponent's pawn chain since objects cannot be prespecified (e.g., the extent of the blockade and whether or not the kings want to get around it must be determined dynamically). PARADISE must reason by linking concepts to create plans that are not implicit in the knowledge base, instead of filling in prespecified slots. To compare actual production rules, MYCIN's actions mention the names of objects while PARADISE's actions involve many variables whose instantiation is not determined until execution of the action.

4. Knowledge Sources and Concepts

When a production in PARADISE matches, it may have an action part which posts various concepts in the data base. The program is able to express and use concepts by dividing its knowledge base into various Knowledge Sources. Each Knowledge Source (KS) provides the knowledge necessary to understand and reason about a certain type of concept. In its simplest form, a KS is a group of productions which knows about some concept and a list of variables such that an

instantiation of the variables represents an instance of this concept. For example, the SAFE KS in PARADISE has two variables, P1 and SQ, and contains productions which know how to make the particular square SQ safe for the piece P1. With this KS, the system can use the concept of a square being safe for a piece in its reasoning, in the expression of plans, and in the communication of discoveries from the tree search. Such concepts are a level above the patterns recognized by the productions themselves.

When a concept has a corresponding KS, PARADISE will eventually treat each instance of that concept which is present in the data base as a subgoal and use the corresponding KS to solve the subgoal in an attempt to produce a plan to realize that instance of the concept. (This may be done by creating other instances of other concepts.) There may also be concepts that have no corresponding KS and are not treated as subgoals. Such concepts are inspected by patterns attempting to match and by the searching routines. When a concept or an instance of a concept corresponds to a KS, it will sometimes be referred to as a goal or subgoal.

Concepts in the data base must contain all the information that will be needed to execute other KSes and to guide the search (see [12]). Among other things, KSes usually expect information on how likely a goal is to succeed and on how threatening a goal is (this is useful in deciding if a sacrifice is warranted). PARADISE stores information about concepts on a list of attribute-value pairs, called an attribute list. For reasons discussed in [12], a particular concept is expressed by giving its name, an instantiation of its arguments, and a list of attribute lists (one for each reason the concept was suggested).

When a production accesses the values of attributes in a concept, it may not care which attribute list the values come from, or it may require that the values for each attribute come from the same list, or it may want the "best" (in some sense) value for each attribute whether the different values come from the same list or not, or it may want values for one attribute to come from lists which have some particular value for another attribute. This accessing of the information in a concept is so complex that it can be looked on as pattern matching in itself. Thus the productions in a KS can be looked upon as matching patterns in the given chess position and matching patterns in different concepts.

Concepts and their accessing are quite complex. Intuitively, the productions in PARADISE are not making deductions, per se, but coming up with ideas that may or may not be right. Thus there are no "facts" that have certain deduced values that can be reasoned with. The complexity of later analysis requires that a production essentially put down "why" it thought of an idea. In this way, other productions can decide if the idea still looks right in light of what is known at that time. Most production systems avoid this complexity because the firing of a production is a deduction which adds a new piece of knowledge to the system. This piece of knowledge is assumed right and the system is not prepared to later decide that it wasn't right to make that deduction.

5. Plans

The goal of the static analysis process in PARADISE is to produce plans. Plans seem to play an important role in the problem solving process of move selection for good human chess players. Alexander Kotov in [6] writes:

... it is better to follow out a plan consistently even if it isn't the best one than to play without a plan at all. The worst thing is to wander about aimlessly. (p. 148)

Most computer chess programs certainly do not follow Kotov's advice. In a knowledge based program, the cost of processing the knowledge should be offset by a significantly smaller branching factor in the tree search. Plans help reduce the cost of processing knowledge by immediately focusing the program's attention on the critical part of a new position created during the search, thus avoiding a complete re-analysis of this new position. Having plans also reduces the branching factor in the search by giving direction to the search. Consider, for example, that most programs may try the same poor move at every alternate depth in the tree, always re-discovering the same refutation. PARADISE can detect when a plan has been tried earlier along a line of play and avoid searching it again if nothing has changed to make it more attractive. Most programs also suffer because they intersperse moves which are irrelevant to each other and work towards different goals. PARADISE avoids this by following a single idea (its plan) down the tree.

Plans in PARADISE can be considered as special instances of concepts since each plan has a list of attribute lists. Instead of a list of instantiations for a set of variables (as in a concept), a plan has an expression in the Plan-Language. The Plan-Language expresses plans of action for the side about to move, called the offense. In general, the offense wants to have a reply ready for every defensive alternative. A plan cannot therefore be a linear sequence of goals or moves but must contain conditional branches depending on the opponent's reply. When the offense is on move, a specific move or goal is provided by the plan. When the defense is on move, a list of alternative sub-plans for the offense may be given. Each alternative begins with a template for matching the move made by the defense. Only alternatives whose template matches the move just made by the defense are tried in the search. PARADISE has the following templates which adequately describe defensive moves in terms of their effects on the purpose of the plan being executed (P and SQ are variables which will be instantiated in an actual plan to particular pieces and squares, respectively): (P SQ) matches when the defense has moved the piece P to the square SQ, (NIL SQ) matches when the defense has moved any piece to SQ, (P NIL) matches when the defense has moved P (to any square), (ANYBUT P) matches when the defense has moved some piece other than P, and NIL matches any defensive move.

The plan for an offensive move can be one of two things: a particular move or a goal. A particular move is simply the name of a piece and a square. Such a plan causes PARADISE to immediately make that move with no analysis of the position other than checking that the move is legal. A goal is simply the name of a KS followed by an instantiation for each argument of the KS. When executing a goal, PARADISE will construct an instance of the KS's concept in the data base by using the given instantiations and the attribute lists of this plan as a whole. The KS will then be executed for this concept-instance (goal). Any plan produced by this execution will replace the goal in the original plan and this modified plan will be used. This process expands and elaborates plans. If executing the KS does not produce a plan then the original plan has failed.



```
((WN N5) (((BN N4) (SAFEMOVE WR Q7)
            (((BK NIL) (SAFECAPTURE WR BR))
              ((ANYBUT BK) (SAFECAPTURE WR BK))))
            ((BN N4) (CHECKMOVE WR Q7) (BK NIL) (SAFECAPTURE WR BQ))))
  ((THREAT (PLUS (EXCHVAL WN N5) (FORK WR BK BR))) (LIKELY 0))
  ((THREAT (PLUS (EXCHVAL WN N5) (EXCH WR BQ))) (LIKELY 0)))
```

Figure 2
a plan produced by PARADISE (white to move)

Figure 2 shows a problem and one of the plans PARADISE's static analysis produces for it. ("WN" means white knight, "N5" means the square N5, etc. It should be obvious to which white rook "WR" refers.) The last two lines of the plan are attribute lists. The first five lines are the Plan-Language expression for the plan which can be read as follows:

Play N-N5. If black captures the knight with his knight, attempt to safely move the rook on Q1 to Q7. Then, if black moves his king anywhere try to safely capture the black rook on R7, and if black moves any piece other than his king, try to safely capture the king with the rook. A second alternative after black captures the knight is to attempt to safely move the rook on Q1 to Q7 with check. Then, if black moves his king anywhere try to safely capture the black queen with the rook.

SAFEMOVE, SAFECAPTURE, and CHECKMOVE are all KSeS in PARADISE. After playing N-N5 and NxN for black in the tree search, PARADISE will execute either the CHECKMOVE goal or the SAFEMOVE goal. Executing the SAFEMOVE goal executes the SAFEMOVE KS which knows about safely moving a piece to a square. This KS will see that R-Q7 is safe and produce this as the continuation of the original plan, causing the system to play R-Q7 with no further analysis. If for some reason R-Q7 was not safe then the SAFEMOVE KS will try to make Q7 safe for the rook (by posting a SAFE concept). If some plan is found, the original plan will be "expanded" by replacing the SAFEMOVE goal with the newly found plan. In general, the newly found plan will contain (SAFEMOVE WR Q7) though it may come after a sacrificial decoy or some other tactic. If posting the SAFE concept does not produce a plan, then the SAFEMOVE goal has failed, and the alternative CHECKMOVE goal will be tried. If it also fails, a complete analysis of the position must be done. If black had answered N-N5 with a king move, it would not match the template (BN N4) and a complete analysis of the position would be undertaken without attempting to make Q7 safe for the rook.

A detailed description of the static analysis process in PARADISE is given in [12]. PARADISE suggests the plan in

figure 2 for two different reasons. One of these two ways of suggesting this plan is briefly traced below. The system begins by posting a THREAT concept since the THREAT KS has productions which look for threats. One production recognizes the white rook could skewer the black king and rook from Q7 so it posts a MOVE concept which says the white rook would be well placed on Q7. A production in the MOVE KS, that recognizes that R-Q7 is unsafe, matches and posts a SAFE concept for making Q7 safe for the white rook. While executing the SAFE KS, one production notices that the black knight blocks the white queen's protection of Q7, and posts a DECOY goal. The DECOY KS has a production which posts a FORCE goal suggesting N-N5 as a move to decoy black's knight. The plan in figure 2 has been built up at this point, although more KSeS are executed (to check for effects) before the final plan is posted.

6. Discussion of plans in PARADISE

By use of conditionals, a plan can handle a number of possible situations which may arise. The most important feature of the Plan-Language is that offensive plans are expressed in terms of the KSeS. This has many advantages. Relevant productions are immediately and directly accessed when a new position is reached. The system has one set of concepts it understands (the KSeS) and does not need to use a different language for plans and static analysis. The system has been designed to make the writing of productions and thus the forming of KSeS reasonable to do. Thus the range of plans that can be expressed is not fixed by the Plan-Language. By forming new KSeS, the range of expressible plans may be increased without any new coding (in the search routines or anywhere else) to understand the new plans. This is important since the usefulness of the knowledge base is limited by its ability to communicate what it knows (e.g., through plans).

Given a number of plans to execute, the tree search must make decisions about which plan to search first, when to forsake one plan and try another, when to be satisfied with the results of a search, and other such things. To make such decisions, it must have information about what a plan expects to gain and why. Such information was available to the productions which matched to produce the plan and must be communicated in the attribute lists of the plan so as to provide many different types of access to this knowledge. Unlike most search-based chess programs, PARADISE must use its information about a plan at other nodes in the tree since it executes a plan without analyzing the newly created positions.

To use the information about plans in an effective manner, PARADISE divides a plan's effects into four different categories which are kept separately (i.e., their values are not combined in any way). These four categories are: THREAT which describes what a plan threatens to actively win, SAVE which describes counterthreats of the opponent a plan actively prevents, LOSS which describes counterthreats of the opponent not defended against and functions the first piece to move in a plan will give up by abandoning its current location (thus providing new threats for the opponent), and LIKELY which describes the likelihood that a plan will succeed. These four categories have emerged during the development of PARADISE. Experience seems to indicate the system must have at least these four dimensions along which to evaluate plans in order to adequately guide the tree search.

Since the values for these categories must evaluate correctly at

different nodes in the tree, they cannot be integers. Instead they (except LIKELY) are expressions in the Threat-Language ([12]) which can describe threats in a sophisticated manner. These expressions are evaluated in the context of the current position and may return different values for different positions. The value of LIKELY is an integer which represents the number of unforced moves the defense has before the offense can accomplish its plan. The most likely plans to succeed (e.g., the plan in figure 2) have a LIKELY of zero (every move is forcing).

The system must quickly recognize when a plan is not working and abandon it before effort is wasted investigating poor lines. On the other hand, it is expensive to completely analyze a position, so the system wants to use its plan as long as possible. To achieve a balance, plans must adequately express the purpose they intend to achieve. Productions must carefully formulate their Plan-Language expressions at the right level of detail using appropriate goals and descriptions of defensive moves. When executing a plan, the system will retrieve a goal or move from the plan to apply to a new position. Care must be taken to ensure that this move or goal is reasonable in the new position. For example, a plan which specifies that any check be played may lack detail and cause many poor checks to be investigated. On the other hand, if the plan specifies an actual move to attack the king, it may be too detailed since this move may no longer check the king in the new position (again a poor line is investigated). Plans should express their purpose by describing how the king should be attacked.

A plan should handle as many replies as possible without causing a re-analysis, but it should avoid suggesting poor moves. The templates for describing defensive moves in the Plan-Language and the various KSeS have been developed to allow PARADISE's plans to accurately express their purpose. The results have been quite satisfying: the productions in PARADISE now create plans which rarely suggest poor moves but which can still be used for as many ply as a human might use his original idea.

7. Using Plans to Guide the Search

Figure 3 depicts a position from [8] and shows the plan PARADISE suggests as best after a static analysis. To show how such plans are used to guide the search, PARADISE's search for this position is sketched below.



```
((WO R7) (BK R2) (CHECKMOVE WR R5) (BK NIL) (ATTACKP BK))
((THREAT (PLUS (WIN BK) (EXCHVAL WO R7))) (LIKELY 1)))
```

Figure 3 (white to move)

The search commences execution of this plan by playing QxPch and producing a new board position. The defense suggests both legal moves, KxQ and K-B1, but tries KxQ first. Since this move matches the (BK R2) template in the plan, PARADISE has ((CHECKMOVE WR R5) (BK NIL) (ATTACKP BK)) as its best plan at ply 3 of the search. Execution of the CHECKMOVE KS produces ((WR R5) (BK NIL) (ATTACKP BK)) as a plan which causes R-R5ch to be played. Black plays his only legal move at ply 4, K-N1, which matches the (BK NIL) template in the plan. Thus PARADISE arrives at ply 5 with (ATTACKP BK) as its plan. Executing the ATTACKP KS posts many concepts, including MOVE and SAFE concepts, and ((WN N6) ((NIL (SAFEMOVE WR R8)) (NIL (SAFEMOVE WN K7)))) is produced as the best plan in this position. After playing N-N6, the position in figure 4 is reached.

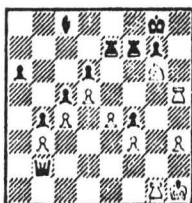


Figure 4

Productions which know about defending against R-R8 suggest ((B8 R6)(B8 Q2)(B8 N2)(BR B4)(BR B3)(BR B1). Black plays BxP first in an effort to save the bishop from the ensuing skewer. At this point the plan branches. Since both branches begin with a null template, they both match any black move at ply 6. Thus PARADISE has two plans at ply 7: (SAFEMOVE WR R8) and (SAFEMOVE WN K7). Before executing a plan for the offense, PARADISE executes the QUIESCENCE KS which looks for obviously winning moves. Here R-R8 is suggested by the QUIESCENCE KS which causes the (SAFEMOVE WR R8) plan to be executed immediately. PARADISE plays R-R8, finds that black is mated, and returns to ply 6 knowing that black's BxP leads to mate. All this has been accomplished without a static analysis; the original plan has guided the search.

At ply 6, black uses KSeS to refute the mating line. No new moves are found since all reasonable defenses have already been suggested. PARADISE has a causality facility which determines the possible effects a move might have on a line of play. Using the tree generated for the mating line, the causality facility looks for effects a proposed move might have (such as blocking a square which a sliding piece moved over, vacating an escape square for the king, protecting a piece which was attacked, etc.). The causality facility recognizes that neither B-Q2 nor B-N2 can affect the mating line found for BxP so they are rejected without searching. Black plays R-B4 next, the causality facility having recognized that this move opens a flight square for the black king. Again PARADISE has both (SAFEMOVE WR R8) and (SAFEMOVE WN K7) as plans at ply 7. Both SAFEMOVE goals would succeed, but R-R8 has a higher recommendation and is played first. Black plays his only legal move at ply 8, K-B2. PARADISE no longer has a plan at ply 9. Before trying a static analysis, it executes the QUIESCENCE KS in an attempt to find an obviously winning move. R-KB8 is suggested and PARADISE immediately plays this move without doing a static analysis. This is mate so PARADISE returns to ply 6 to look for other

defenses. Both R-B3 and R-B1 are tried and both are quickly refuted by playing R-R8 from the SAFEMOVE goal of the original plan, K-B2, and R-KB8 from the QUIESCENCE KS. The search then returns to ply 2 and tries K-B1 in answer to QxPch. The template in the original plan does not match K-B1 so there is no plan at ply 3. However, the QUIESCENCE KS quickly suggests Q-R8 and PARADISE returns from the search convinced that QxPch will mate.

PARADISE's plans are not always so accurate but space prevents presentation of a longer search. The above analysis uses about 130 seconds of cpu time on a DEC KL-10. It goes to a depth of 9 ply while creating only 21 nodes in the tree. Because of the guidance provided by the original plan, no static analysis was performed except on the original position. By comparison, the TECH2 program (an improved version of TECH [5]) at a depth setting of 6 discovers that it can win material with QxPch although the horizon effect hides the mate from it. For this analysis, TECH2 uses 210 seconds of cpu time and creates 439,459 nodes by making legal moves (it also makes and retracts 544,768 illegal moves).

8. Measuring PARADISE's performance

To aid in evaluating performance, PARADISE was tested on positions in [8]. This book contains tactically sharp positions from master games which are representative of tactical problems of reasonable difficulty. PARADISE's knowledge base was developed by writing productions which would enable the program to solve, in a reasonable manner, fifteen chosen positions from among the first 100 in [8]. The 85 positions not chosen were not considered during this development. This development process produced one version of the program, called PARADISE-0. Six positions were then picked at random from the remaining 85 and accurate records were kept on the work involved in getting the program to solve these reasonably. The version of the program which solves all 21 positions is called PARADISE.

PARADISE's performance on a problem is classified into one of three categories: 1) problem solved as is (possibly with a minor bug fix), 2) problem not solvable without a change to the program or a major change to the knowledge base, or 3) problem solvable with a small addition to the knowledge base. Category 3 helps measure the modifiability of the knowledge base. It is meant to include solutions which require no changes whatsoever to the program and only a small addition to the knowledge base. Small means that it takes less than 20 minutes total of human time to identify the problem and write or modify one production which will enable PARADISE, with no other changes, to solve the problem in a reasonable way (i.e., no ad hoc solutions).

Of the six positions chosen at random, two were in category 1, three were in category 3, and one was in category 2. This latter one inspired the only program changes between PARADISE-0 and PARADISE. These results speak well for the modifiability of the knowledge base, but shed little light on the generality of the program. To better test generality, PARADISE has been tested on the first 100 positions. These positions are divided into 5 groups of 20 and Reinfeld claims an increase in difficulty with increase in group number. (Eight endgame positions were eliminated, so the groups actually have 18, 19, 18, 20, and 17 positions.) PARADISE is considered to solve only the problems in category 1, while PARADISE-2 solves both category 1 and 3 problems. PARADISE would

become PARADISE-2 simply by leaving in the productions written to solve problems in category 3. The following table shows what percentage of these problems can be solved by PARADISE, PARADISE-2, CAPS [1], TECH [5], CHESS 4.4 [11] running on a CDC 6400, and a human rated as class A [1]. PARADISE was limited to 45 minutes of cpu time per problem while the other programs were limited to 5 minutes.

	PARADISE	PARADISE-2	CAPS	TECH	C-4.4	HUMAN
Group I	78%	100%	67%	78%	94%	89%
Group II	68%	95%	74%	84%	95%	95%
Group III	94%	100%	61%	61%	78%	94%
Group IV	70%	95%	50%	40%	70%	80%
Group V	65%	94%	41%	47%	76%	53%
All 92	75%	97%	59%	61%	83%	83%

There were 20 problems in category 3 in the first 100, but only 13 productions were written to solve them. In two instances, an already existing production was modified. In five instances the same production solved 2 different category 3 problems. This is a strong indication that the productions being written are fairly general and not tailored to the specific problem. PARADISE already exhibits more generality in this domain than programs like TECH and CAPS. The figures for PARADISE-2 show that these problems do not push the limits of the expressibility of the production language nor the ability of the program to control the tree search. These results indicate that the generality of PARADISE is reasonable and that the modifiability of the knowledge base is excellent.

It should be noted that PARADISE uses considerably more cpu time and produces trees with considerably fewer nodes than either of the other programs. The following statistics were compiled for the 89 problems which PARADISE-2 solved in these 92. PARADISE averaged five and one-half minutes of cpu time per problem (it ranged from 19 seconds to 1958 seconds), and generated an average of 38 nodes per problem (ranging from 3 nodes to 215). On the average, PARADISE spent 53% of its time calculating primitives. There were an average of 3.7 static analyses done per problem, and less than 9% of the nodes generated had static analyses done on them. This shows that the plans do a good job of guiding the search.

9. Comparison to Plans in Robot Problem Solving

Plans are used in many domains of AI research, especially robot problem solving. ABSTRIPS [9], NOAH [10], and BUILD [4] are examples of planning systems in robot problem solving environments. Plans in these systems are very different from those in chess so a summary of these differences is more appropriate than a detailed comparison. These differences are explained in greater detail in [12]. Robot planning is frequently done in abstracted spaces which have been defined, and which omit details. Such an approach is much more difficult in chess, at least with mankind's current understanding of the game. It is not clear how to define abstracted spaces. Small details are very important in chess and cannot be readily ignored.

In the robot planning systems, the effects of an action are well defined. The program can easily and quickly determine the exact state of the world after an action. In chess, moves (actions) may subtly affect everything on the board. Chess plans cannot make many assumptions about the state of the world in the future, but must describe the expected features of

new states in the plan itself and then test for these features while executing the plan.

In most robot planning systems, tests for having achieved the goal or the preconditions of an action are trivial and give well defined answers. In chess, there may be only very subtle differences between a position where a particular action is good and positions where the same action is wrong. It is also hard to know when a goal has been achieved, since there is always a chance of obtaining a larger advantage (except when the opponent has been checkmated). Thus in chess it is necessary for the plan to provide a considerable amount of information to help in the making of these decisions. This is done in PARADISE through the attribute lists in plans.

The number of things PARADISE can consider doing at any point in a plan is about an order of magnitude larger than the number of things most robot problem solvers usually contend with. There are an average of about 38 legal moves in a chess position (see [2]). When planning farther in the future than just the first action, the chess planner has many more choices than the 38 legal moves. The inability to make assumptions about future states often prevents mentioning actual moves in a plan. Instead a description of the intent is needed, and the number of such descriptions is much larger than the number of legal moves that might be made. A robot usually has a much smaller number of possible actions, so the two types of plans explode at very different rates.

The robot planners have the flavor of establishing a sequence from a small number of well-understood operations until the correct order is found, while chess planners have more a flavor of "creating" the correct plan from the many possibilities. For this reason, a system which produces good chess plans needs a large amount of knowledge and non-trivial reasoning processes to produce plans. Plans in PARADISE differ from those in robot problem solvers by specifying powerful knowledge to be applied in new situations. Plans essentially give PARADISE a perspective to use when analyzing a new position.

10. Comparison to Plans in Chess

Pitrat's program [7], which solves chess combinations, is the most important example of the use of plans in chess. The language Pitrat uses for expressing plans has four statements with the move statement and modification statement being the most important. The move statement specifies that the piece on one particular square should move to another particular square. It may also be specified that the move must be a capture. The modification statement describes a modification to be made to a particular square. This can be one of four things: removal of a friend, removal of an enemy, moving a friend to the square, or getting an enemy to move to the square.

Plans in PARADISE provide much more flexibility in expression than Pitrat's move statement. In PARADISE, types of moves other than captures can be specified (e.g., safe moves). Also, particular squares do not need to be named in PARADISE since it can use a variety of goals to express the plan. The important square to move to may change depending on the opponent's move so it is not always possible to specify such a square in advance. Pitrat's modification statement is a goal which the system tries to accomplish. These modifications are too simple to express the purposes behind their suggestion. For example, the system may want to decoy the black queen to

make a square that it protects safe for white. Pitrat would express this goal as removing an enemy from the black queen's location which does not express the purpose of the plan. It will work well for the winning combination but will also allow many wrong moves since the queen may be decoyed to a square from which she can still protect the square in question. PARADISE would specify that the black queen must be decoyed to make the particular square safe, and only decoys which remove the black queen's protection would be considered. PARADISE attempts to express the purpose of each plan while Pitrat's plans express side effects that will happen if the purpose is accomplished (e.g., a square becoming vacant). Unfortunately, the same side effects may also happen when the purpose is not accomplished, causing the system to waste effort searching poor lines.

Plans in PARADISE have two more major advantages over the plans in Pitrat's system. First, PARADISE has conditionals which allow specification of different plans for different replies by the opponent. The advantages of this are obvious: the system can immediately try the correct plan instead of searching an inappropriate plan and backtracking to correct itself. Second, PARADISE's plan language is modifiable. Simply by writing new productions, new goals and concepts can be created for expressing plans and the system will automatically understand these new plans. This property is necessary for any system that wishes to extend its domain or incrementally increase its expertise. Significant additions to Pitrat's plan language would seem to require a major programming effort.

It should be noted that Pitrat's program processes nodes much faster than PARADISE and therefore can handle larger trees. It may be the case that programs will obtain better performance by using larger trees and less sophisticated plans, but the use of knowledge is only starting to be investigated by programs such as PARADISE and it is too early to draw conclusions.

11. Summary

PARADISE exhibits expert performance on positions it has the knowledge to understand, showing that a knowledge based approach can solve some problems that the best search based programs cannot solve because the solutions are too deep. The knowledge base is organized into KSeS which provide concepts for PARADISE to use in its reasoning processes. These concepts are at a higher level than the ideas produced by simply matching patterns. PARADISE's knowledge base is amenable to modification. Production rules (and therefore KSeS) can be quickly written and inserted in the knowledge base to improve system performance without adverse affects.

The concepts provided by the KSeS are used to create plans during static analysis. The concepts communicate enough information that they can be rejected after being posted. The same concepts are used to express plans for guiding the search, and the KSeS execute the plans during the search. By communicating plans down the tree, PARADISE can understand new positions on the basis of its analysis of previous positions. By having particular goals in mind when looking at a new position, the system quickly focuses on the relevant part of the data base and avoids using all its patterns. Plans in PARADISE express their purpose fairly well, without being too general or too specific. Because of the successful communication of knowledge through plans, PARADISE is

able to solve a number of chess combinations with small trees and without a depth limit on its search.

One of the major issues in creating concepts is their generality. If the concepts are too general then too many details are lost, and the system does not have the necessary facts to do certain reasoning. If not general enough, the system leans toward the extreme of needing a production for every possible chess position. The concepts in PARADISE have been constructed to be as general as possible without sacrificing expert-level performance to the loss of detail. The tradeoff of generality and specificity in concepts is closely related to the tradeoff of search and knowledge. With much specific knowledge the use of knowledge becomes expensive, but fewer mistakes need to be corrected in the search. With more general knowledge, analysis is cheaper but more time must be spent correcting mistakes in the search. PARADISE uses a mix of search and knowledge which involves more knowledge and less search than previous programs which include chess middle games in their domain. This type of research should strive for generality within the confines of its commitment to knowledge-based expert performance.

REFERENCES

- [1] Berliner H, "Chess as problem solving: The development of a tactics analyzer", Unpublished doctoral thesis, Carnegie-Mellon University, 1974.
- [2] Charness N, "Human Chess Skill", Chapter 2, in *Chess Skill in Man and Machine*, Frey, P. (Ed), Springer-Verlag, 1977.
- [3] Davis R, "Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases", AIM-283, C. S. Department, Stanford University, 1976.
- [4] Fahlman S E, "A planning system for robot construction tasks", *Artificial Intelligence* 5, pp. 1-49, 1974.
- [5] Gillogly J, "The technology chess program", *Artificial Intelligence* 3, pp. 145-163, 1972.
- [6] Kotov A, *Think Like a Grandmaster*. Dallas: Chess Digest, 1971.
- [7] Pitrat J, "A chess combination program which uses plans", *Artificial Intelligence* 8, pp. 275-321, 1977.
- [8] Reinfeld F, *Win At Chess*, Dover Books, 1958.
- [9] Sacerdoti E D, "Planning in a hierarchy of abstraction spaces", *Artificial Intelligence* 5, pp. 115-135, 1974.
- [10] Sacerdoti E D, "The nonlinear nature of plans", Stanford Research Institute Technical Note 101, January 1975.
- [11] Slate D, Atkin L, "CHESS 4.5- The Northwestern University chess program", Chapter 4, in *Chess Skill in Man and Machine*, Frey, P. (Ed), Springer-Verlag, 1977.
- [12] Wilkins D, "Using patterns and plans to solve problems and control search", forthcoming doctoral thesis, Computer Science Department, Stanford University.