THE UNIVERSITY *of* EDINBURGH

VALIDATION OF MACHINE-ORIENTED STRATEGIES IN CHESS ENDGAMES

by

Timothy B. Niblett

Ph.D. University of Edinburgh 1982

Acknowledgements

Thanks are due to Donald Michie, my supervisor, for his constant support and encouragement, and also to Ivan Bratko, Michael Clarke and Danny Kopec for many useful discussions. I am grateful to Max Bramer for many detailed and helpful comments on an earlier version of the thesis.

Table of Contents

1. Overview 4

1.1 Introduction 4
1.2 Outline of thesis 11

2. The Game Theoretic model of Chess 15

2.1 Introduction 15
2.2 Solving subgames of chess 17
2.2.1 Computational complexity of solutions 24
2.3 Human performance 29
2.4 The Turing-Shannon paradigm 30
2.4.1 Defects of the paradigm 32
2.5 Knowledge-based programs for the middlegame 34

3. Previous Work on Chess Endgames 38

3.1 Introduction 38
3.2 Chess theory 40
3.3 Bramer's work 42
3.4 Huberman's model 45
3.5 Advice Language 1 49

4. An automatic proof method for AL1 strategies 56

4.1 Introduction 56
4.2 The KRK advice table 57
4.3 Generalised database construction 60
4.4 Formal definition of procedures 66
4.4.1 Better goals only 66
4.4.2 Move constraints at ply one 66
4.4.3 Holding goals 67
4.4.4 Comparison predicates 67
4.5 Proof Procedures 68
4.6 Mechanical proof of Bratko's lemmas 70
4.7 Mechanical Proof 76

5. The King and Rook vs. King and Knight Ending 83

5.1 Bratko's original advice table 83
5.2 Errors in the advice table 89
5.3 An analysis of a textbook description of KNKR 90
5.4 Correction of the KNKR table 97
5.5 The performance of the new KNKR table 99
5.5.1 Separated positions 99
5.5.2 King and knight together 103
5.6 Defects of AL1 106
5.7 Using Plans 116

6. An advice strategy for KPK 135

6.1 Introduction 135
6.2 The play of King and Pawn vs. King 138
6.3 The Advice Strategy 147
6.4 Search 158
6.5 Implementation 162
6.5.1 A new search technique 163
6.5.2 The Algorithm 164
6.5.3 Reduction of search 167
6.6 Program Details and an example of play 169
6.7 Conclusion 175

7. Summary of results 177

8. REFERENCES 179

I. Appendices 185

I.1 Construction of a KRKN database 185
I.2 Backing up Algorithms 191
I.3 Correctness proof for the KPK strategy 198
I.4 Description of the KNKR tables 218

1. Overview

'If there be nothing new, but that which is
Hath been before, how are our brains beguiled,
Which labouring for invention bear amiss
The second burthen of a former child!'

Shakespeare

1.1 Introduction

This thesis is concerned with the validation of chess endgame strategies. It is also concerned with the synthesis of strategies that can be validated. A strategy for a given player is the specification of the move to be made by that player from any position that may occur. This move may be dependent on the previous moves of both sides. A strategy is said to be correct if following the strategy always leads to an outcome of at least the same game theoretic value as the starting position. A more formal definition of these terms is given in chapter 2.

We are not concerned with proving the correctness of programs that implement the strategies under consideration. We shall be working with knowledge-based programs which produce playing strategies, and assume that their concrete implementations (in POP2, PROLOG etc.) are correct.

The synthesis approach taken attempts to use the large body of heuristic knowledge and theory, accumulated over the centuries

by chessmasters, to find playing strategies. Our concern here is to produce structures for representing a chessmaster's knowledge which can be analysed within a game theoretic model.

The validation approach taken is that a theory of the domain in the form of the game theoretic model of chess (described in more detail in chapter 2) provides an objective measure of the strategy followed by a program. Our concern here is to analyse the structures created in the synthesis phase. This is an instance of a general problem, that of quantifying the performance of computing systems. In general to quantify the performance of a system we need,

- A theory of the domain.
- A specification of the problem to be solved.
- Algorithms and/or domain-specific knowledge to be applied to solve the problem.

A simple example to illustrate all three of the above items is shown in Figure 1.1. Here the domain theory consists of axioms for lists, written in first order logic. The axioms are a version of the Peano axioms for arithmetic, adapted to a different data structure. The specification of the problem is given in terms of a definition of concatenation, again written in first order logic. The logic programming school has shown that in many cases, including this one, the specification of the problem can be used to provide efficient solutions given a simple interpreter for

Figure 1.1

Axioms for lists

$$\text{list}(\text{nil})$$

$$\forall x\ \forall y(\text{list}(x.y) \Leftrightarrow \text{element}(x) \wedge \text{list}(y))$$

$$\forall x\ \forall y(\text{nil} \neq x.y)$$

$$\forall x\ \forall y\ \forall x'\ \forall y'(x.y = x'.y' \Leftrightarrow x = x' \wedge y = y')$$

Definition of Concatenation

$$\forall x(\text{concatenate}(\text{nil},x,x))$$

$$\forall x\forall y\forall z\forall w\forall w'(\text{concatenate}(x.y,z,w) \Leftrightarrow \text{concatenate}(y,z,w') \wedge w = x.w')$$

Induction Schema

$$P(\text{nil}) \wedge \forall x\ \forall y(\ P(y) \Rightarrow P(x.y)\ ) \Rightarrow \forall z P(z)$$

first order logic.

We can use this example, and similar examples can be found within computer science, as a paradigm for our investigations of knowledge representation in chess. In our case, although the theory of the domain and specification of the problem are tractable, it is a great challenge to provide a framework for the representation of chessmasters' knowledge which allows both a solution of problems within feasible computational bounds and the application of game theory to show that the strategies we produce are correct.

It may be asked why it is necessary to validate endgame strategies. Why not just build systems to synthesise them, and determine performance from their observed behaviour (and failure)? Why not repeat this in an iterative cycle until the desired performance is reached? There are several arguments against this position.

There is a combinatorial explosion with this approach. The search space of possible programs is too large to search unaided without some help, such as methods for quantifying the performance of the programs.

The synthesis of strategies in a knowledge-based system can require a large amount of domain knowledge. The coding of such

knowledge is a time consuming and tedious task. One way of easing the burden of supplying such knowledge is to automate the procedure by starting with a partial solution and inducing the extra knowledge needed to produce correct strategies. This process is simpler if we have procedures by which the strategies produced by this extra knowledge can be automatically verified.

Finally, formalisation of how we represent knowledge is an essential first step in the generalisation necessary if we are ever to classify differing representation techniques. If this is not done there is a danger that people will continually 're-invent the wheel'.

WHY USE CHESS ENDGAMES?

Chess is a good domain within which to investigate knowledge representation issues because it is not only highly complex but also a knowledge-rich intellectual activity. If we were able to construct a system to play at grandmaster level using techniques of planning, reasoning and knowledge representation we could solve similarly complex problems in other domains with analogous properties.

Any attempt to program chess by knowledge-poor methods runs

into the combinatorial explosion. The knowledge-based approach combats this with knowledge of the domain. This domain knowledge is easily available for chess as there is a vast body of knowledge developed by chessmasters over hundreds of years. In other domains this theory is often lacking and has to be painstakingly developed. Non-chessplayers are often surprised at the extent of chess theory. Even for endgames with only kings and pawns on the board, the standard textbook (Averbakh & Maizelis 1974) has several hundred pages and only gives a cursory summary of many results. The incorporation of this knowledge into a playing program is a challenge because of its variety, incompleteness and heuristic nature.

Attempts to write algorithmic programs incorporating extensive domain knowledge also seem inherently limited. Zuidema (1974) reports very great difficulty in writing a correct program (in the game theoretic sense) to play KRK.

We have at our disposal a powerful tool for the analysis of chess - modern game theory. Chess is a finite, zero sum, two person game of perfect information. Hence it can be reduced to normal form (see chapter 2 for a precise definition). Thus the game always has a value and from any legal position there is a deterministic strategy for each side which guarantees at least the game-theoretic value. Such strategies are called minimax strategies. This model is not directly applicable to the full

game of chess because of the combinatorial explosion of search. However chess is decomposable into subgames and the simpler ones are susceptible to exhaustive analysis based on the game theoretic model. This can then be used to analyse the performance of knowledge-based programs in these subgames.

To summarise the above arguments, using chess endgames as our field of research has the following advantages:

1. They present an intellectually challenging domain, which conventional algorithmic programming techniques cannot easily solve.

2. There is a large body of chess theory available, largely of a pragmatic nature. If we can encode this theory into knowledge-based systems we will learn a great deal about knowledge representation techniques.

3. Game theory can be successfully applied to simple endgames, providing an analytic tool with which to analyse the performance of various representation schemes.

In addition,

1. We can compare program play with that of human chessmasters. They are highly skilled in their art and objective measures of their performance are available.

2. We can compare the play of knowledge based programs with that of conventional (knowledge-poor) chess playing programs when tested in particular endgames. These have been intensively developed in the past 15 years and their behaviour is well understood.

3. Chess strategies are easy to program since the underlying rules are simple. This allows the experimenter to concentrate on the knowledge representation issues involved.

1.2 Outline of thesis

The tool used in the investigation of validation methods for endgame strategies is the AL1 system (Michie 1976). AL1 was chosen as a system for endgame play designed to facilitate formal proof of the correctness of its play. A method for proving lemmas about AL1 strategies by exhaustion in a computationally feasible manner is described, and a worked example given for the endgame of King and Rook vs. King (KRK). We then examine the applicability of this method to the more complex endgame of King and Rook vs. King and Knight (KRKN), where strategies for the knight's side are considered. It is argued that although these validation techniques are applicable to KRKN it is not easy to synthesise strategies that are correct. Furthermore any attempt at such a synthesis leads to a system that is both difficult to modify incrementally and conceptually opaque. Results new to chess theory are presented for the endgame.

We consider methods for the synthesis of strategies in a hierarchical manner, which eases the problem of synthesising correct strategies. A worked example for the endgame of King and Pawn vs. King is given. A provably correct strategy is presented. The hierarchical nature of the synthesis technique allows the

incremental incorporation of extra knowledge in an easily controllable fashion.

In chapter 2 the game-theoretic model of chess is discussed. We see how it can be used to produce a complete lookup table for simple endgames. We discuss the notions of correctness and optimality of strategies in terms of this model, and how this relates to human performance. It turns out that even in such endgames human play is almost invariably sub-optimal, and often incorrect. This throws light on the way chess theory is adapted to human computational abilities. The ability of programs based on the Turing-Shannon paradigm to play correctly is discussed. Finally we discuss knowledge-based programs for the middlegame in chess.

In chapter 3 various models that have been implemented to represent chess textbook knowledge are reviewed from the point of view of their amenity to correctness proofs of their strategies. We consider the work of Bramer (1977) who produced a minimax-optimal strategy for the king and pawn vs. king endgame. This model eschews search in favour of pattern matching. We then consider the work of Huberman (1968). She used a model for endgames based on the idea of a <u>forcing tree</u> which facilitates formal proof of correctness. Finally we describe the AL1 system which is based on Huberman's approach but with a more flexible control structure, enabling a wider range of problems to be

solved.

Chapter 4 focusses on methods of automating the proof of AL1 strategies. Using the endgame of king and rook vs. king we show how a system to act as a proof checker of user-produced lemmas can be developed. This is used to give an alternative proof for the correctness of a KRK strategy, developed by Bratko. A more direct technique is also developed to directly prove correctness. This technique has the effect of compiling the strategy, producing precomputed moves from every position.

In chapter 5 a detailed investigation is made of AL1 strategies for the king and rook vs. king and knight ending. A lookup table for the endgame was constructed and with it results new to chess theory were discovered. Against our expectations it was found that correct play in the endgame is hard even for chessmasters. Errors in the play of an initial version of the advice strategy led Bratko (1979) to develop a second version which is also shown to be incorrect. We analyse the reasons for this failure and suggest that a richer model of the domain is needed. Suggestions are made respecting this model.

Chapter 6 gives a worked example of an advice strategy for the king and pawn vs. king endgame. The strategy is provably correct. A novel technique of implementation is described that considerably eases the representation of tactical knowledge for

the domain, and allows a richer control structure for the search.

In chapter 7 a brief summary of the results of the work is given.

2. The Game Theoretic model of Chess

2.1 Introduction

Game theory, introduced by Borel, generalised by von Neumann and given definitive form in 'The Theory of Games and Economic Behaviour' (von Neumann and Morgenstern 1944), allows many otherwise intractable problems to be solved within a very general framework. The basic results of the theory - the reduction of games to normal form, the minimax theorem etc. are directly applicable to chess, indeed this was done in (von Neumann and Morgenstern 1944). In the terminology of game theory, chess is a finite, zero sum, 2-person game of complete information.

It is interesting to note that Zermelo (Zermelo 1912) seems to have been the first person to prove that every legal chess position has a well-defined value. However his approach is less useful than the game-theoretic one since it does not suggest a practicable method of solving sub-games of chess.

Chess is characterised by a set of rules which for every possible position specify legal moves to other positions. It is possible to move from this intensional definition to an

extensional one where the game is seen as a tree, whose nodes correspond to possible positions, and whose edges correspond to possible moves. Given this tree we can define the set of playing strategies for White and Black. A strategy for White specifies the first move and for each possible reply a continuation, and so on. In general for each sequence of choices $m_1, \ldots, m_{2k}$, $k > 0$ it specifies a choice $c(m_1, \ldots, m_{2k})$ for the $(k+1)$st move for White. The idea of a forcing tree which we meet later is a generalisation of this notion of strategy.

It is possible to specify chess as a table of strategies for each side. This table is finite (very large) and every choice of a strategy for each side has a value, win, loss or draw. Let the set of strategies for White be $W_1, \ldots, W_n$, for Black $B_1, \ldots, B_m$. The minimax theorem for finite games of perfect information, such as chess, states that there is a pure minimax strategy for each side. There is some White strategy W_i and a corresponding Black strategy B_j such that if the payoff of W_x against B_y is M_{xy} then

$$M_{ij} = \min_y \{M_{iy}\} = \max_x \{M_{xj}\}$$

Although every finite 2-person game has a minimax value there is not necessarily a pure minimax strategy. In this case the minimax value is obtained only by a mixed strategy which randomly chooses one of the available strategies. The proof given by von Neumann for the existence of pure strategies in games such as chess defines the order of a subtree of the game tree. The order of a

subtree is the maximum length of the game, with best play by both sides, from the root of the subtree. The notion of order is essentially that of minimax optimal depth which we shall use in solving endgames. Indeed von Neumann's proof is constructive and the construction similar to that of databases for endgames.

2.2 Solving subgames of chess

Chess is decomposable into subgames. A subgame, broadly speaking, is the full game of chess restricted in some way. The restrictions in which we are interested are those which specify which pieces must lie on the board. More stringent conditions are possible, for example certain pieces may be restricted to lie on certain squares. A formal definition of subgame, suitable for our purpose, is given below. To fully compute these subgames some simplification is necessary in our model. The game theory model requires that the (k+1)st move made by White depends on the previous 2k moves of the game, so any node in the tree effectively consists of the position of the pieces plus the previous history of the position. This history can be condensed to the following,

1. Which side is on move.
2. Castling status.
3. En passant status.
4. Information necessary for determining threefold

repetition of position, which results in a draw if requested by the side on move.

5. Information relevant to the 50-move rule. This rule states that if no pawn has moved or piece been captured in the last 50 moves a draw may be claimed by the side on move. We shall assume that such situations are always drawn, which reduces chess to a finite game.

The model we use ignores all these factors except the first. In the endgames we consider it is assumed that castling is not allowed and that none of the positions has been encountered before. These restrictions do not significantly alter the problem, but by allowing us to consider identical arrangements of the pieces as identical positions they provide us with a finite model.

The model has been chosen for its convenience in describing the solution of various endgames. We assume that each side has pieces:

$W_1, \ldots, W_{nw}$ for White

$B_1, \ldots, B_{nb}$ for Black

The model is a bipartite directed graph G(V,E) satisfying:

$$V = V_W \cup V_B \qquad V_W \cap V_B = 0$$

$$E \subseteq V \times V$$

$$(v_1, v_2) \in E \Rightarrow v_1 \in V_W \text{ and } v_2 \in V_B$$

$$\text{or} \quad v_1 \in V_B \text{ and } v_2 \in V_W$$

The elements of V are positions. Positions are tuples of the form: $(w_1, \ldots, w_{nw}, b_1, \ldots, b_{nb})$,
where the w_i and b_j are in {0, .., 63}.

and $v \in V \Rightarrow w_i \neq w_j$, $i,j \leq nw$, $i \neq j$

and $w_i \neq b_j$, $i \leq nw$, $j \leq nb$

and $b_i \neq b_j$, $i, j \leq nb$ $i \neq j$

that is no two pieces are coincident.

$E(v_1, v_2)$ iff there is a legal move from v_1 to v_2
The precise definition of E depends on the subgame of chess under consideration.

Note that some illegal positions are included in the graph, for instance those where the two kings are adjacent. We can purge these by defining a new graph G(V',E') where

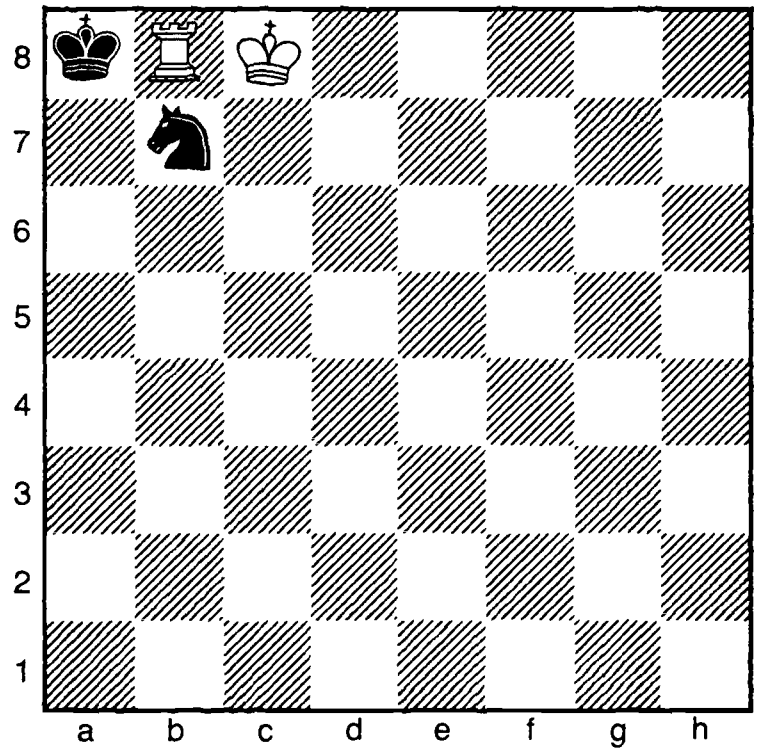
$v \in V' \Leftrightarrow v \in V$ and $\exists v_1 (E(v_1, v))$

E' = E restricted to V' x V'

This is a dangerous procedure since it excludes some positions we may wish to include, an example is shown in Figure 2.1. In general it seems preferable to leave problems of legality to the definition of legal move. A legal move from v to v_1 always has the property that in v_1 the side on move in v is not in check.

With the above definitions we can now define a <u>solution</u> to the subgame given by G(V,E). The idea here is that given a set of

Figure 2.1



Black to move

This position has no predecessor in the KRKN space, yet the position may legally occur. White's last move could have been PxN = R check.

Black-to-move positions (I_0), we can for every positive integer j, find a set I_j of positions from any one of which White can force a position in I_0 within j half-moves (ply). The choice of White as the side to force a position in I_0 is arbitrary.

Let $S(v) = \{ v_1 \mid E(v,v_1) \}$, the successors of v.

Let $S^{-1}(v) = \{ v_1 \mid E(v_1,v) \}$, the predecessors of v.

A solution for G(V,E) with respect to I_0 V_B is the family of sets I_j, $j \in \mathbb{N}$ satisfying

$$I_{2i+1} = \{ v \mid v \in S^{-1}(I_{2i}) \} - \bigcup_0^{2i} I_j$$

$$I_{2i+2} = \{ v \mid v \in S^{-1}(I_{2i+1}) \wedge S(v) \subseteq \bigcup_0^{2i+1} I_j \} - \bigcup_0^{2i+1} I_j$$

The following consequences of this definition can easily be proven

1) $\forall i,j \quad I_i \cap I_j = 0, \quad i \neq j$

2) $\exists j \leq \mid I_j \mid \neq \emptyset$ and $I_{j+1} = \emptyset$ from 1) and the fact that $\mid \bigcup_0^N I_j \mid$ is monotonically increasing with N and V is finite.

We now define a <u>strategy</u> for White to achieve goal g from position v as any tree T satisfying:

1. g holds at all leaves of T

2. v is in T

3. $v' \in V_W$ is in T => g(v') or $\exists v''$ S(v') s.t. $v'' \in T$

4. $v' \in V_B$ is in T => g(v') or $\forall v'' \in S(v')$, $v'' \in T$

5. T is finite

Note that a position v' in T can occur more than once on any branch so that T is not a subgraph of G(E,V). This notion of strategy is closely analogous to the one we saw above. Intuitively a strategy exists for some goal from position v (with either White or Black to move) iff White can force a goal position from v whatever the opponent's play. We now see that the notion of a strategy and a solution to the graph link up.

Theorem 2.1

There is a strategy T from v for g iff $v \in \bigcup_0^\infty I_j$ where

$I_0 = \{ v \mid g(v) \}$

Proof

Sufficiency

We show that the root of T is in $\bigcup_0^\infty I_j$.

From the definition of strategy every subtree of T is a strategy for g.

We proceed by induction on the length of the maximal branch in T, which is finite by Konig's lemma.

case 0: T = v and g(v) => $v \in I_0$

case n+1: By the induction hypothesis all elements of

the subtree(s) of T below the root are in $\bigcup_{0}^{N} I_j$

for some N. By definition of the I_j the

root of T is in some I_j.

Necessity

We construct a tree T recursively from v.

We describe the construction of T for each j where $v \in I_j$.

case 0: T = v

case 2n+1: $j = 2n + 1$ then $\exists v' \in S(v)$ s.t. $v' \in I_{2n}$.

We can construct a strategy T' from

v'. Append T' to v to get T.

case 2n+2: $j = 2n + 2$. $\forall v' \in S(v)$ $v' \in \bigcup_{0}^{n} I_{2i+1}$

We can construct a subtree for each v', append the

subtrees to v to get T.

We say that a strategy is optimal if for every branch of the tree with nodes $v_0, \ldots, v_m$ with $v_0 \in I_{n0}, \ldots, v_m \in I_0$ we have $n_0 > \ldots > 0$. The strategy created in the second half of the above proof is optimal and so we have the result that if there is a strategy there is an optimal strategy. Such strategies are commonly called minimax optimal strategies, though any strategy for a goal is minimax in the game theoretic sense.

2.2.1 Computational complexity of solutions

The solution we have described is amenable to machine computation for endgames with only a few pieces since the problem space can be represented as a relatively small (order 10^6 - 10^7) graph. The solution is expressed as a lookup table with the minimax-optimal value of each position stored. It is simple to generate optimal play from this lookup table. Given a position in I_n generate all successors, the optimal move is the move to successor v' which belongs to the lowest (for White) or highest (for Black) ranked I_j; if a position v' is not in any I_j its rank is considered to be plus infinity. Full details of the creation of such a lookup table for the endgame of King and Rook vs. King and Knight are given in Appendix 1. The construction in the proof above can be directly implemented.

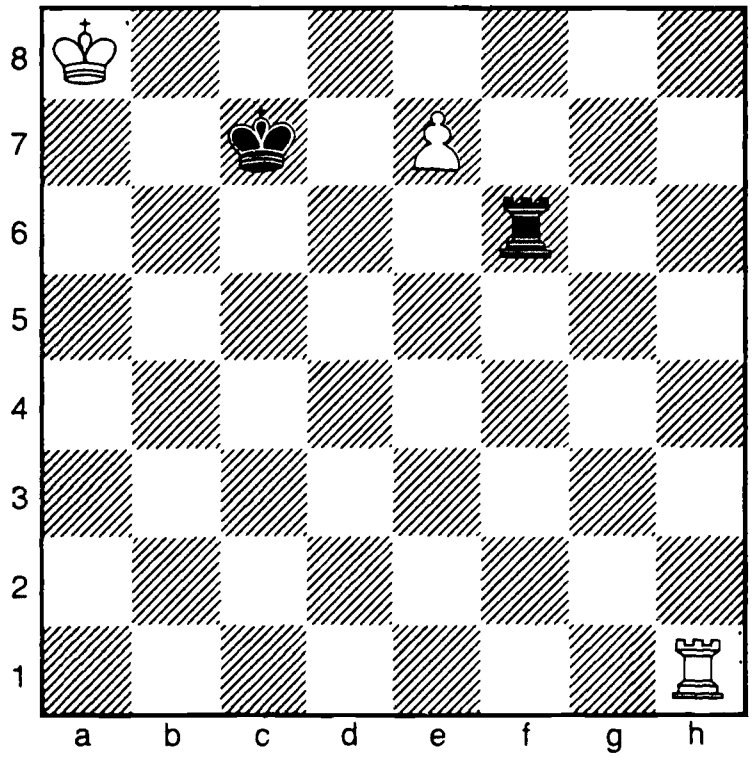
The first database seems to have been created by Thomas Strohlein (1970). The largest collection of such databases to be constructed is probably that of Ken Thompson of Bell labs, who computed databases for most of what he calls the interesting four piece endings. Programming these endings is an extremely delicate task, with a multitude of possibilities for error. The only practical way of testing the correctness of any one program seems

to be independent generation of more than one database. One of the best validated databases by this criterion is for the King and Rook vs. King and Knight ending. Three independent versions by Thompson, Gams and Niblett are all in agreement: specifically Thompson's was checked position by position with Niblett's, while Gams' was checked by comparing the distribution of losses over depth of loss with that of Niblett.

Databases have also been constructed for KRPKR and KPQKQ (Arlazarov and Futer 1979), endings with 5 pieces. Their solution for KRPKR is however flawed because it ignores the possibility of transposition into the ending KRNKR which is sometimes necessary. Figure 2.2 shows a KRPKR position where White must promote to a knight to win. Five piece endings with a pawn can be decomposed into subgames by pawn rank and file, this is not possible for KRNKR which therefore needs very much more space for computation. The present limits to fast central memory make the computation infeasible.

Let us briefly consider the computational cost of creating a database. The cost can be split into 2 parts, that of initialising all immediately won positions and that of performing the backing up process. If we assume a separate inspection of every position in the Black-to-move space the initialisation process will take time proportional to the number of White-to-move positions. Backing up involves generating all legal predecessors

Figure 2.2



White to move

White wins with e8 = N, other moves draw at best.

It is not known how the Arlazarov-Futer database classifies this position: their omission of the KNKR subgame precludes correct classification.

of every position lost for Black or won for White. This is done only once. The cost associated with backing up is therefore proportional to the number of lost positions for both Black and White plus the cost of searching, at each depth of loss, for positions lost at that depth. Figure 2.3 summarises this information. The amount of space required for the generation will be proportional to the size of the space of Black and White-to-move positions.

The main alternative to this process of backing up is to perform a depth-first minimax search from every position. This method is rather harder to evaluate. It is usually assumed that we desire the computation to be space-minimal, which involves using store proportional to the maximum depth of search. There is a problem in specifying exactly what this depth should be however. If we know beforehand the depth at which a loss occurs we can limit the search to this depth. If not we can bound the depth only by the fifty move rule (with the assumption that it will always be invoked). In Figure 2.3 it is assumed that we know the maximum necessary search depth beforehand. From this comparison we can see that the database method is faster in time, for a given number of pieces, but requires exponentially more store. We are able to exploit this tradeoff at a favorable point for simple endgames.

Figure 2.3

Cost of database generation

$$\text{Time} \propto G(N_W + N_B) + I \times N_B \approx (2G + I)64^{Np}$$

$$\text{Space} \propto N_W + N_B \approx 2 \times 64^{Np}$$

Cost of minimax search

$$\text{Time} \propto D^B (N_W + N_B) \approx 2 \times D^B \times 64^{Np}$$

$$\text{Space} \propto D$$

Np = no. of pieces

N_W = no. of WTM positions

N_B = no. of BTM positions

B = average branching factor

D = max. depth of loss in ply

G = time taken to generate a set of successor positions

I = time taken to check whether a position is a depth 0 loss

2.3 Human performance

Surprising results have been obtained when chessmasters have played against databases over the board. At Toronto in 1978 two International Masters, one of them Hans Berliner a former world correspondence chess champion, were pitted against Ken Thompson's KRKQ database. They were unable to demonstrate a win with the stronger side against the program's optimal play. One reason for this failure was that the optimal strategy employed by the program involved a separation of the king and rook which is counter to accepted chess theory. Separation of the 2 pieces is considered risky because of the danger of losing the rook to a pin, fork or skewer. A similar phenomenon was found to occur in the KRKN endgame during our research in Edinburgh.

The explanation for this flaw in chess theory seems to lie in the cognitive strategies employed by the chessmaster. He plays by generating plans which avoid lines of play too complex to analyse in terms of the plans he can generate in the position. A study, described later, of a chess expert's performance with the KRKN database indicates that with increased knowledge the chessmaster will be able to solve problems in the domain. An extension to chess theory is possible which will then cover the endgame adequately.

2.4 The Turing-Shannon paradigm

We now turn our attention to the exploitation of the game theory model of chess by chess playing programs. In theory we can evaluate any position by searching the game tree, in minimax fashion, to termination. The Turing-Shannon approach reduces the amount of search necessary by introducing an evaluation function which provides a cheap estimate of the minimax value of a position. If the function were a perfect estimator the maximum depth of search necessary would be only one ply. In practice it is imperfect, and we find empirically that the greater the depth of search, the greater the accuracy of estimation. Despite its intuitive appeal and the empirical findings it is not easy to prove that a limited-depth minimax search is more effective than no search at all. Recent work done independently by Beal (1982) and by Bratko and Gams (1982) has shown that accuracy of evaluation increases with depth only when neighboring positions in the game tree have mutually dependent values. It would be interesting to measure this dependence for simple endgames such as King and Pawn vs. King.

Such is the basis of almost all chess programs for the full game of chess. The question we shall address here is: what can we say about the performance of such programs? A great deal of research and practical experimentation has been done within the Turing-Shannon paradigm in the past 20 years. The performance of

tournament programs has improved greatly. The latest generation of programs are rated in the top 1 percent of registered chessplayers. The major factors contributing to this increase in performance since the early 1960's are probably:

1. An increase in the speed of tree generation. BELLE (Condon & Thompson 1982) the current (1980) world champion evaluates over 150,000 positions per second.
2. The use of the alpha-beta tree pruning algorithm allied to the technique of progressive deepening, and other value-preserving search heuristics.
3. A better understanding of where to terminate search.
4. Experience with the fine tuning of the evaluation function.

Considerable understanding has been achieved of the behaviour of various tree searching algorithms. Shannon's original (1950) paper suggested that a forward pruning of the search tree by considering only a certain number of possible moves, selected by an evaluation function, would produce better results than a search without forward pruning. One surprising result to date has been that this selective search has not performed as well as the full width alpha-beta search. Selective search seems to need a great deal of knowledge to guide it, and this requirement conflicts with the need for speed of position evaluation. Those who have so far attempted to pursue a knowledge-based approach within the Turing-Shannon paradigm have found that the restrictiveness of the model frustrates their efforts. A further reason is that moves rejected by forward pruning are precisely those that are likely to

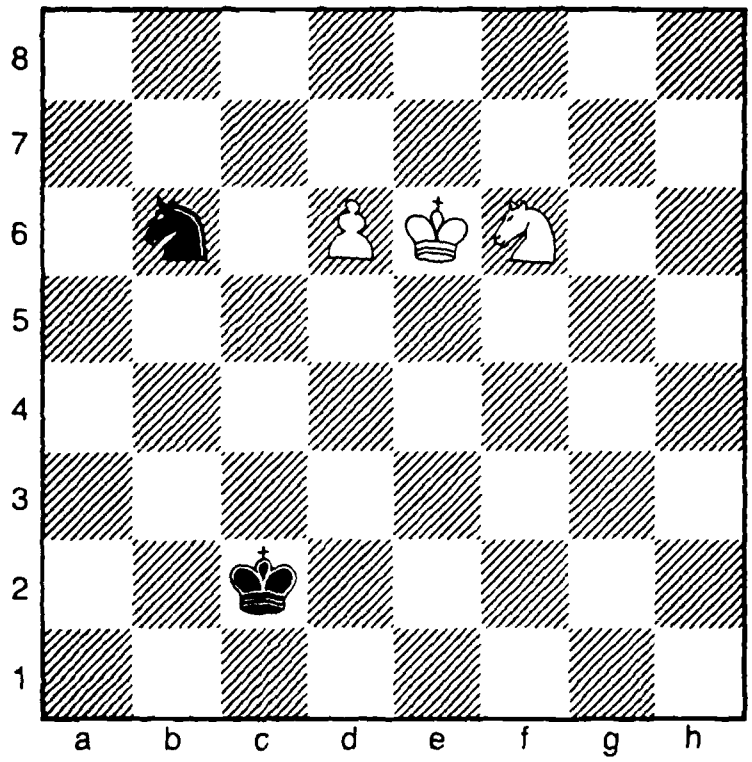
be pruned by the full-width alpha-beta algorithm, especially when equipped with additional tricks such as the 'killer heuristic' (Birmingham and Kent 1980).

2.4.1 Defects of the paradigm

In light of the remarkable advance in playing strength of chess programs in the past 5 years it seems foolhardy to predict that they will never beat human masters or grandmasters. Indeed in the field of lightning chess where each player has five minutes to complete the game programs are already a serious challenge to masters. Despite this the paradigm has inherent defects which seem insurmountable.

Berliner (1974) observed that programs are oblivious of events beyond their lookahead horizon. This results in the 'horizon effect' which has both a positive and a negative form. The positive effect occurs when a small gain occurring within the horizon is chosen over a larger gain which occurs outside the horizon. The negative effect occurs when the program sacrifices material or position to delay a larger loss which is inevitable anyway. An example of the positive effect can be seen in Figure 2.4. To avoid the horizon effect involves communication of results between different branches of the tree, indeed a

Figure 2.4



White to move

The positive horizon effect

If White advances the pawn the game is drawn despite the win of a piece. The correct move is Nd5 to stop the Black knight covering the pawn's advance. The main variation is,

1. Nd5 Na4 2. Nb4+ K any 3. Na6 any

4. Pd7 and queens in at most 2 moves.

communication of an abstraction of the results. Again this only seems possible within a knowledge-based approach such as discussed below.

The evaluation function used is incapable, because of its 1-dimensionality of producing a consistent plan of play throughout the game. The problem can be thought of in terms of the traditional hill-climbing analogy. It is possible that at some stage in the game the best plan involves making moves which are not optimal according to the evaluation function. The program cannot see over the hill.

2.5 Knowledge-based programs for the middlegame

The acknowledged defects of the Turing-Shannon approach, in particular the difficulty of achieving a more formal ability to quantify its performance inevitably leads to an inability to incrementally improve the performance of programs in any systematic fashion, or even to predict whether changes will in fact improve performance. This has led people to try a knowledge-based approach to chess playing programs. A great deal of work has been done on programs that play tactical middlegames. Tactical play is chosen for the ease with which problems can be posed and evaluation of the program carried out. We shall consider the work of Wilkins (1980), which builds on work done by Berliner

(1974) and Pitrat (1980).

Wilkins' program PARADISE uses a knowledge base of about 200 productions, some of which are very complex, to discover plans for White in tactical middlegame positions and to drive a small tree search to discover which plan is best. The program generates a set of primitives from the initial position. These discover threats, potential checks and various aggressive and defensive relations between pieces. These primitives are accessed by the knowledge base to produce an initial set of plans. The production system architecture is similar to that of the HEARSAY system, the productions being grouped into knowledge sources which communicate by means of a global blackboard. The information passed between productions can be extremely complex and at more than one level of generality. PARADISE has the following noteworthy features:

1. It abstracts from the board situation and forms plans from high-level descriptors.

2. Tree search is limited (usually between 10 and 100 nodes) because the search only considers plans both for the attacking and defensive side. Only a limited number of counterplans are considered for the opponent. Plans do not consist merely of move sequences, but can consist of more abstract elements such as; 'make square c7 safe for the rook'.

3. Communication is possible between nodes of the tree. If a plan fails the reasons for its failure can be communicated back up the tree and corrective action taken. This is the causality facility. Its power stems from the ability to describe failure at a sufficiently abstract level.

4. The tree search does not use minimax search but a version of the B* search described by Berliner (1974). This has 2 advantages. Firstly the tree searching is

not subject to any depth limit, allowing solutions at arbitrary depths in the tree. Secondly interesting strategies such as 'disprove rest' which are not available to minimax programs can be used.

Unlike the minimax programs Wilkins' approach seems amenable, in principle at least to a more formal understanding. Ideally we would like to be able to do the following,

1. Define the space of possible plans/counterplans. Once this is done we can try and show that the space of plans is complete. If it is not we would like to incrementally modify it until it is complete.
2. Show that the search strategy used is capable of finding any of the plans in this space.
3. Show that the search terminates when no such plan exists.

With even a partial ability to do the above it would be possible to incrementally improve the performance of the program, and to permit the application of techniques of inductive learning to aid in this improvement. Unfortunately the actual implementation does not allow a characterisation of the program on these lines. Wilkins does however give details of its performance. He gives three positions from a test set of 92 where the program failed to find a solution. It failed largely because the set of available plans was inadequate. On no position tested did the program produce an unbounded search, though this seems to have been the result of careful tuning rather than an inherent robustness.

Although empirical testing of this sort is important and

necessary, it does not provide an adequate guarantee of performance for two reasons. First, there is no guarantee that anomalous behaviour will not occur in any given position. This is a phenomenon familiar to the users of any large software system, unexpected errors occur in situations seemingly similar to ones where the system behaves correctly. Second, the ability to formally specify the behaviour of a complex system such as Wilkins' allows a prediction of where it will fail so that knowledge can be added incrementally. If this is impossible then the system cannot be developed beyond a certain level of competence. It is very important to be able to find the 'exceptions' which always occur. We shall meet this problem again as we proceed. It is central in any attempt to create correct programs (in the game-theoretic sense) for chess endgames.

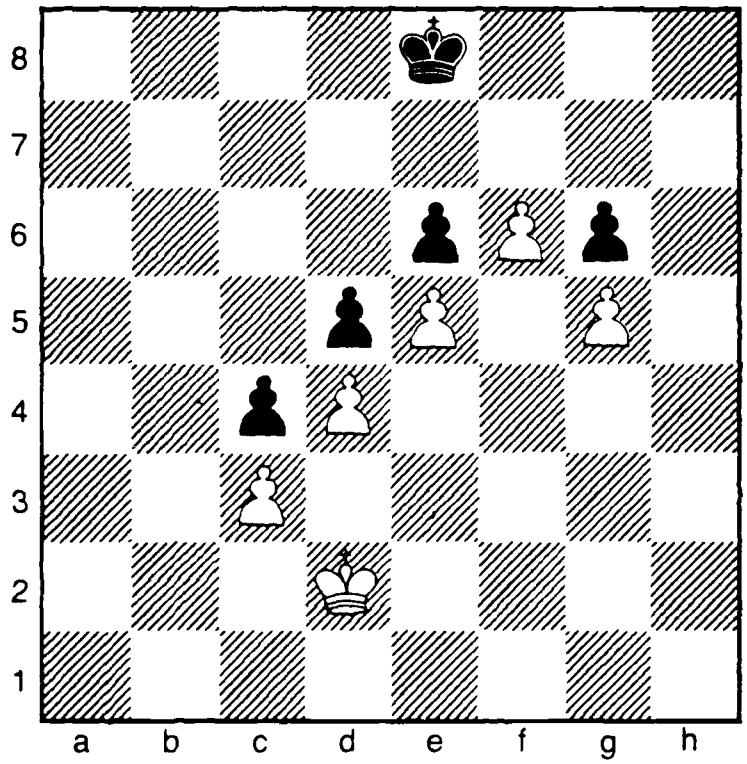
3. Previous Work on Chess Endgames

3.1 Introduction

In chapter 1 we saw that to specify the performance of a program in some domain it is desirable to have a description of the domain, expressed as a theory of the domain. This theory should be capable of effective implementation. The direct approach of implementing the game-theoretic model via minimax search plus evaluation function does not work well for endgames. One reason for this can be seen by studying Figure 3.1. The position is won for White, and most chessplayers will find the solution very quickly. The White king advances to capture the e6 pawn. Black cannot prevent this since his king is tied to the White pawn on f6. A program using a pure minimax search will not easily find this solution as the pawn is not captured until ply 17.

One approach is to provide a richer framework of chess knowledge. We shall consider how to provide this framework while allowing a demonstration of the correctness (in the minimax sense) of a program's play. One advantage of studying chess is the presence of a large body of chess theory, gathered over the past centuries. Before continuing it is worth surveying this theory briefly.

Figure 3.1



White to move

White can move the king to e6, capturing Black's pawn and winning.

3.2 Chess theory

The game of chess is normally divided into three parts, the opening, the middlegame and the endgame. The division is not hard and fast, there are many positions which do not fall easily into any one category, but the three are analysed differently by theory.

The themes underlying modern opening theory are pawn formation and piece placement, with the aim of steering the game towards a complex and unbalanced struggle. Openings are categorised by the pawn structures they exhibit, and their different approaches to key factors such as control of the centre of the board. The evaluation of lines of play is in no way fixed. It is comparatively rare for top flight masters to play the same lines as 30 years ago. Characteristic of opening theory is a dialectic process of criticism and rehabilitation of key lines of play. Very little work has been done on the development of a general theory of any precise nature for the opening.

In the middlegame a distinction is drawn between tactical and strategic play. Tactical play is of a relatively short range - usually less than half a dozen moves, with well defined goals of

material or positional gain. Strategic aspects are longer range, often reaching into the endgame. A further distinction can be drawn by considering the types of planning involved in the two kinds of play. Strategic planning involves consideration of long term features of a position, such as pawn structure. The plans tend to be at a fairly high level of abstraction, for example the order in which the pieces are positioned is rather unimportant. Tactical planning involves short term features of position such as pins. The ordering of moves in a tactical plan is usually precise.

There is often a predominance of tactical considerations in the middlegame which seems to explain the relatively better performance of chess programs in this area. The endgame is characterised by reduced material and a predominance of strategic considerations. Specific lines of play are less important. The emphasis is on high level ideas such as pawn structure, critical squares, the ability of pieces to cover long-term threats in more than one area of the board etc. The necessity for high level planning in the endgame makes it an ideal area in which to study knowledge representation. We emphasise three areas of particular interest.

1. The concepts used in our theory of the domain.
2. The planning process. Methods for defining the space of plans, and methods for finding plans.
3. Ways of using plans to produce correct play and of proving that the plans produce correct play. This can involve the need to describe the system formally.

In the remainder of this chapter we consider work that has been done in chess endgames, with special emphasis on the correctness (in the game-theoretic sense) of strategies. We consider the work of Bramer (1977), Huberman (1968) and the AL1 system developed by Michie (1976) and others.

3.3 Bramer's work

Bramer (1977) aimed at developing a model to represent endgame knowledge from textbooks satisfying two criteria:

1. The model should be natural and of a complexity commensurate with a chessplayer's view of the task.
2. The model should be capable of refinement to produce correct play without losing the properties of naturalness and simplicity.

The long term goal of his research was to produce a model capable of automatic refinement, hence the need for a uniform, non-procedural representation. He departs from approaches based on tree search in favour of a static model using equivalence classes of positions. Bramer appears to have been the first person to produce a validated strategy for KPK using a knowledge-based approach.

The position space for the ending under consideration is

divided into equivalence classes, exhausting the space. Each class is intended to represent a type of position conforming to a simple pattern found in the textbook. These patterns are based on relative positions of the pieces. The classes are totally ordered by a relation *better*.

The algorithm for play consists of generating all legal moves from a White-to-move position (where White is the stronger side). The successor position with the highest class value is chosen to move to. In the event of several successor positions falling into the same highest class the tie is broken by considering the *associated functions* of that class. If the tied positions also tie when tested against the first associated function, they are tried against the second. If there is still a tie after considering all the associated functions a move is chosen at random from the tying positions.

Bramer produced models for the endings KRK and KPK. The programs underwent a process of iterative refinement, their performance being measured against databases for the endings. We must distinguish three types of 'correctness' possible for these programs. This classification is due to Bramer.

1. *Value-preserving* From any won position the program moves to a won position. From any drawn position the program moves to a drawn position.

2. *Correct* The strategy followed by the program is minimax in the sense of chapter 2.

3. Optimal The strategy followed by the program is minimax-optimal in the sense of chapter 2.

Value-preserving play is not necessarily correct or optimal. Optimal play is always correct and correct play is value-preserving. Notice that given a database for an ending it is always possible to see whether a particular move is optimal or value-preserving, while correctness is a global property in relation to the information stored in a database.

Bramer produced a value-preserving algorithm for KPK consisting of 19 classes. He refined this further to produce an optimal model of 38 classes (Bramer 1980b). Later research showed that the 19 class model was incorrect. It was refined to produce a correct 20 class model (Bramer 1981).

The demonstrations of optimality and correctness Bramer has given were achieved by generating the move played by his program from every position in the domain. To prove optimality these moves were checked against a database for the ending. To prove correctness he used a technique similar to Algorithm One of Chapter 4 below. The techniques used by Bramer are not directly applicable to validating strategies synthesised by a program which uses search more than 1 ply deep, such as AL1.

3.4 Huberman's model

A rather different model was used by Huberman (1968). Huberman's programs were the first explicitly designed for endgame play. She studied three endgames, KRK, KBBK and KBNK. Her aim was to study the process of translating textbook heuristics into knowledge structures in a program.

The model used was a forcing tree, a generalisation of the game theory idea of strategy. For each endgame two functions better and worse are defined. A forcing tree from position p is defined to be the subtree of the game tree rooted in p satisfying:

1. p is in T
2. At any leaf position q of the tree better(p,q)
3. At any BTM position q all successors of q are in T or q is a leaf
4. At any WTM position q in the tree at least 1 successor of q is in the tree.

Notice that all terminal positions are BTM, and that better and worse take 2 arguments, the root node and the current position in the tree. The worse predicate is not used in the definition of a forcing tree but in the pruning of the search space when trying to find a forcing tree.

Search is done breadth first, two heuristics are used in the

search:

1. Redundant branch cutoff. This heuristic is incomplete.
2. Killer heuristic. Reorders move consideration.

Incomplete here means that the search will sometimes fail to find a forcing tree when one exists. Huberman claims that this never occurred in practice.

Huberman's main concern was finding suitable definitions of better and worse, both to accurately reflect play given in the book and to produce acceptable search characteristics.

DEFINITION OF BETTER

The <u>better</u> predicate is defined by means of <u>stages</u>. In a manner similar to Bramer's the space of positions is divided into equivalence classes, or stages, with an ordering between them satisfying the following constraints:

1. stage(q) > stage(p) => better(p,q) { q is better than p }
2. better(p,q) => stage(q) $\geq$ stage(p)

The relation in 2) is not strict because the stages represent a rather broad division of the position space, and the strict inequality would lead to unmanageable search. Stages are further restricted in that Black can never force a return to a lower stage. White may however be forced to do so.

To provide finer discrimination each stage may have a measure associated with it. Two positions within a stage can be ranked with respect to better by the measure. In this case we have:

1. better(p,q) /\ stage(p) = stage(q) = n => measure (p) > measure (q)
   n n

DEFINITION OF WORSE

Even with the refinement to better provided by the measures, which can considerably limit the depth of search, search is unacceptably large due to the high branching factor. The worse predicate is provided to limit the breadth of search. Worse satisfies the relation:

worse(p,q) <=> stage(q) = 0 \/ (stage(q) = stage(p) = n
/\ measure (p)<measure (q))
n n

The worse predicate is used to prune White's moves during search. Any move which leads to a 'worse' position is pruned.

To program an ending Huberman made preliminary definitions of better and worse in terms of stages, derived directly from textbook descriptions of play. Fine tuning was then done, in an ad hoc fashion, to reduce search to an acceptable level while maintaining program correctness. Program play was close to the textbook examples studied.

Huberman made a serious attempt to produce a model which

could be proved correct. She claimed to be able to show that her programs for the three endings studied could be proven to force checkmate from any won starting position. She sketches a proof for the KRK ending. To show that the program can win from any won starting position she tried to show:

1. That the program can force positions q which are better than any given starting position p
2. That the process need only be repeated a finite number of times from any given starting position before checkmate is achieved

It is clear, from the definition of better that 2) follows from 1), since all measures are finite integers, each stage has an associated measure and there are only a finite number of stages. This approach needs some refinement to prove that the program mates within 50 moves. The sketch of a proof that Huberman gives for KRK indicates that a full proof would be excessively tedious. It would seem to be a major undertaking to provide such a proof for the other, more complex endings.

Huberman could have taken a different approach in the KRK ending however, which though less general is much simpler. She could prove that the program always forces a decrease in the squad function used to measure the degree of constraint of the Black king. This approach is not directly generalisable to the other two endings she studied.

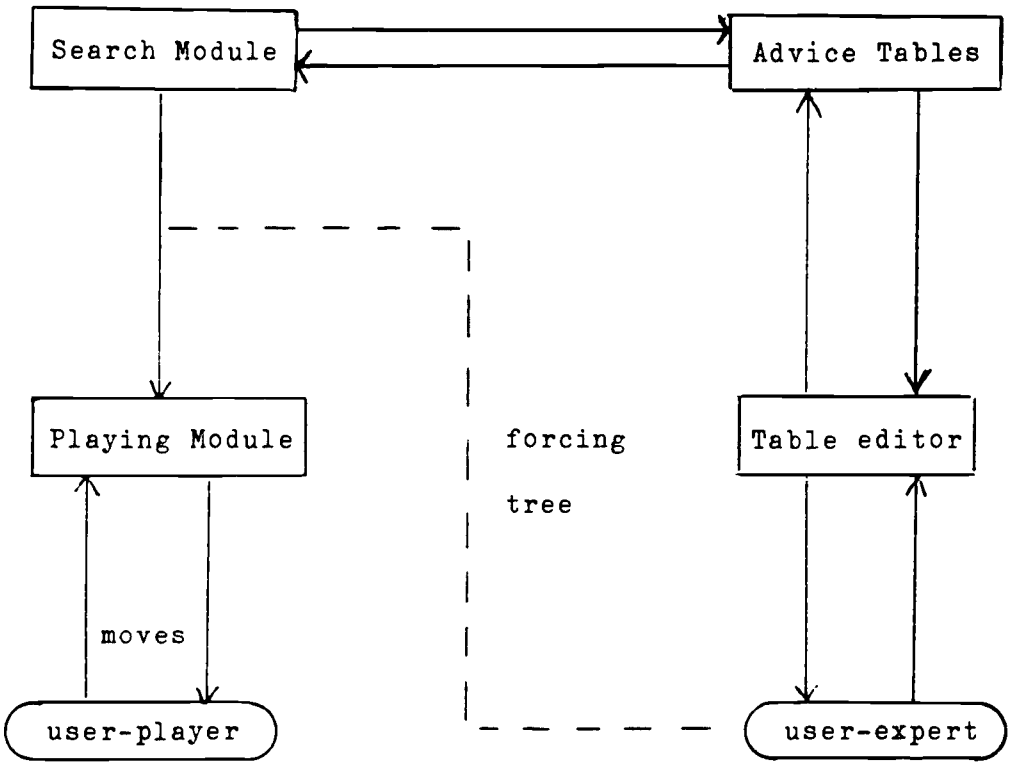
3.5 Advice Language 1

Advice Language 1 (AL1) (Michie 1976, Bratko and Michie 1980), developed by Michie and a class of graduate students at the University of Illinois, is a tool for the implementation of expert systems to play chess endgames. The AL1 system, equipped with a suitable knowledge base can play a variety of chess endgames. The system borrows ideas from Huberman but is more flexible and general.

Figure 3.2 depicts the top-level structure of the AL1 system. The problem solver (search module) constructs from the input position a forcing tree or playing strategy. The playing module executes this strategy over the board.

The knowledge of the system is on 2 levels. At the top level is a set of advice tables, organised via a master table. The master table takes the input position and routes it to the appropriate advice table. In the experiments performed to date this routing merely involves detecting the endgame to which a position belongs and passing this position to the relevant advice table . In theory the matching could be more complex.

Figure 3.2



Top level structure of the AL1 system.

The broken line shows that the user-expert can examine the forcing tree produced by the search module.

An advice table is a set of rules or productions, organised in a form of decision table. The general form of an advice table is shown in Figure 3.3. Rule matching is done by evaluating certain predicates defined by the advice table designer. Execution of a rule produces a predetermined advice list which is passed to the search module. This module attempts to construct a forcing tree, starting with the piece of advice at the head of the advice list and working down. If the list is exhausted without a forcing tree being constructed, failure occurs. An advice table for the KRKN ending is shown in Fig 5.1a.

The lower level of knowledge consists of a collection of predicates, used for matching rules in the advice tables and for evaluation of nodes in the forcing trees. Each endgame has its own collection of predicates.

The chess expert's knowledge is coded into pieces of advice, the building blocks of an advice table. A piece of advice is defined as a sextuple (UBG,UHG,TBG,THG,MC,DB) where

UBG is Us-to-move (UTM) better goal
UHG is UTM holding goal
TBG is Them-to-move (TTM) better goal
THG is TTM holding goal
MC is move constraint
DB is depth bound

A piece of advice A is satisfiable from position P (written s(A,P)) if there is a forcing tree for A rooted in P. A forcing

Figure 3.3

Predicates	Rules: R 1	R j	R m
P 1	v 11	v 1j	v 1m
P i	v i1	v ij	v im
P n	v n1	v nj	v nm
	L 1	L j	L m

The v_{ij} are in the set {'Y','N','-'}.

The L_j are advice lists.

A rule R_j matches a position pos, if for each i ($i < n$)

v_{ij} is '-'

or

if P_i is true in pos then v_{ij} = 'Y'

or

if P_i is false in pos then v_{ij} = 'N'.

A rule R_j is selected in position pos if j is the least integer such that R_j matches pos.

tree is any minimal tree (ordered by inclusion) satisfying:

1. P is in T

2. At every UTM terminal node either UBG holds or, with no better goal, the node is at the depth bound. At every TTM terminal node either TBG holds or, with no better goal, the node is at the depth bound

3. With the exception of the root node, every UTM node satisfies the UTM holding goal and every TTM node satisfies the TTM holding goal

4. The length of the longest branch in T does not exceed the depth bound

5. From every UTM node there is at most one branch. From every TTM non-terminal node there are as many branches as legal moves

6. The move from the root node satisfies the move constraint

This definition of forcing tree is considerably more complex than Huberman's. The use of a depth limit is the major change. It allows for the possibility of a piece of advice being unsatisfiable, without having to worry about unbounded search. It also enables a more efficient depth-first implementation of search. In addition there are now two sets of better and holding goals. One set is applied at Us-to-move positions (the UBG and UHG), the other at Them-to-move positions (the TBG and THG). Huberman applied holding goals only at what we are calling Us-to-move nodes, and better goals only at Them-to-move nodes.

The following algorithm summarises the action of the Search and Playing modules.

1. Read initial position P0. Set current position P=P0. Set MODE=weak.

2. Set FT = empty tree.

3. Consult the Master Table and assign to variable T that Advice Table whose precondition is satisfied by P. If no precondition is satisfied then exit.

4. If MODE = weak or there is no non-terminal node in FT equal to P then:

    a. If the post-condition of T is satisfied in P then goto 2.

    b. Find in table T the first rule R (in left to right order) such that P matches R. Assign to L the advice-list corresponding to R.

    c. Try in turn for pieces of advice A1, A2, etc. in L to generate a forcing tree satisfying the piece of advice, until a piece of advice Ai is found or L is exhausted. In the latter case exit.

    d. Set FT = forcing tree for Ai and P. If Ai has no better-goal then set MODE=weak else MODE=strong.

5. Find in FT the node corresponding to the current position P. Output the move leading from that node in FT and update P by this move.

6. Read the opponent's move and update the current position P by this move

7. goto 4.

Two endgames, KRK and KNKR, have been programmed with the AL1 system. These implementations are the subject of the next two chapters.

The KRK table, written by Ivan Bratko (1978) follows almost exactly the strategy implemented by Huberman. Bratko succeeded in

providing a formal proof of the strategy, considerably simpler than the one attempted by Huberman. Huberman's approach is more general in some ways than Bratko's.

The KNKR ending is far more complex, and distinguished in that here we are trying to maintain the draw rather than win. In this case, since we have no formalisable notion of what constitutes a drawn position, a different approach to demonstrating correctness must be taken.

4. An automatic proof method for AL1 strategies

4.1 Introduction

Bratko (1978) has provided a proof of his Advice Table for King and Rook vs King (KRK) by non-mechanical means. The burden of this proof rests on five lemmas. In this chapter I demonstrate a method of proving these lemmas mechanically, and also a more direct proof that the Advice Table plays correctly. This second proof allows an exact quantification of the program's performance.

The method I use is applicable to a wide range of AL1 advice tables , including Bratko's original table for the KRKN ending. It can act not only as a check on the table's correctness but also as a 'compiler' for the table, enabling the table's play to be stored in a lookup table. It provides an extensional definition of the table's play.

4.2 The KRK advice table

The pieces of advice in Bratko's advice table for the KRK ending are shown in Figure 4.1. The table consists of only one rule, which matches all positions in the KRK space. The action part of the rule consists of five pieces of advice. In any position the highest ranking piece of advice satisfiable is used to generate a forcing tree. This forcing tree is played through to the end. This is play in strong mode, as opposed to weak mode where only the first move in the forcing tree is played. If the forcing tree does not lead to checkmate the table is re-entered and another forcing tree generated.

The strategy followed by the table is very similar to Huberman's. The use of five pieces of advice rather than one, makes the program considerably simpler however.

Bratko's proof uses a technique similar to that of Floyd (1967) for proving program correctnes, and relies on our ability to assert that if a piece of advice A is satisfiable, after execution of a forcing tree for A its better goals will hold in the resulting position.

Figure 4.1a

	Advice				
	MATE	SQUEEZE	APPROACH	KEEPROOM	DIVIDE
TTM better goals					
MATE	Y	-	-	-	-
NROOMLT	-	Y	-	-	-
NOT REXPOSED	-	Y	Y	Y	Y
OKCSNMDLT	-	-	Y	-	-
OROKTKDN	-	Y	-	Y	Y
OROKTKDN OR LPATT	-	-	Y	-	-
OKORNDLE	-	-	-	Y	-
ROOMGT2 OR NOT OKEDGE	-	-	Y	Y	-
TTM holding goal					
NOT STALEMATE	Y	Y	Y	Y	Y
Max. Search Depth	3	1	1	1	5
Move Constraints					
CHECK THEN ANY	X				
RMOVE		X			
KDIAG THEN KMOVE			X	X	

The pieces of advice in the KRK table

Figure 4.1b

Key to KRK table

MATE	Their king is checkmated.
NROOMLT	The room to which their king is confined by our rook has decreased.
REXPOSED	Their king can attack our rook before our king can defend it.
OKCSNMDLT	Our king has approached the critical square to help the rook. The critical square is that square diagonally adjacent to the rook within the area to which the Black king is confined.
LPATT	The 3 pieces form an L-pattern: kings in opposition and the rook next to our king and at a knights jump distance from their king.
OROKTKDN	Our rook divides the 2 kings vertically or horizontally.
OKORNDLE	The distance between our king and our rook has not increased.
ROOMGT2	The room to which their king is confined contains more than 2 squares.
OKEDGE	Our king is on the edge.
RMOVE	A move by our rook.
KMOVE	A king move.
KDIAG	A diagonal king move.

4.3 Generalised database construction

The basis of the proof system is a technique for finding, in a computationally feasible time, all positions in the domain of KRK satisfying a given piece of Advice. It is also possible to find the moves that may be played by the Advice Table from these positions.

Although a method of exhaustive evaluation of the move played in each position was used by Bramer to demonstrate the correctness of a strategy in the endgame of King and Pawn vs King, it is impractical to do this using the AL1 system. For KRK alone, allowing an average of 30 seconds per move this could take up to one month of CPU time on the DEC 10 machine used in these experiments. The amount of search necessary from each position is far greater than in Bramer's case. For the King and Rook vs King and Knight endgame, with a position space of 1.4 million legal WTM positions, exhaustive evaluation of this kind is infeasible. Such search seems to be necessary for the KRKN ending, for at least two reasons. Firstly search is needed to avoid tactical losses. If this is not done the search must be compiled into the patterns used which greatly increases their complexity. Secondly it may be possible to specify fairly simple better goals, which can be achieved within 4 or 5 ply, while being almost impossible to

specify such predicates for a 1 ply search.

The method I use to overcome this problem is a generalised form of the method of solving subgames of chess by database creation, described in chapter 2 (section 2.2). Given a piece of advice A the method finds all positions in the KRK domain in which A is satisfiable. In addition it finds all the moves that may be played by the table from these positions.

The method of chapter 2 finds all positions from which checkmate or material capture can be forced by one side in a given number of moves, this number being the minimax optimal depth of the position with respect to the subgame of chess being considered. A forcing tree, in the sense of AL1, does something similar. If a forcing tree can be generated from some position P we are assured that certain better-goals can be satisfied within a given depth. A forcing tree is more complex since,

1. Holding goals and move constraints have to be satisfied throughout the tree.
2. Better goals and holding goals are predicates on more than one position, requiring a limited amount of communication within the tree.

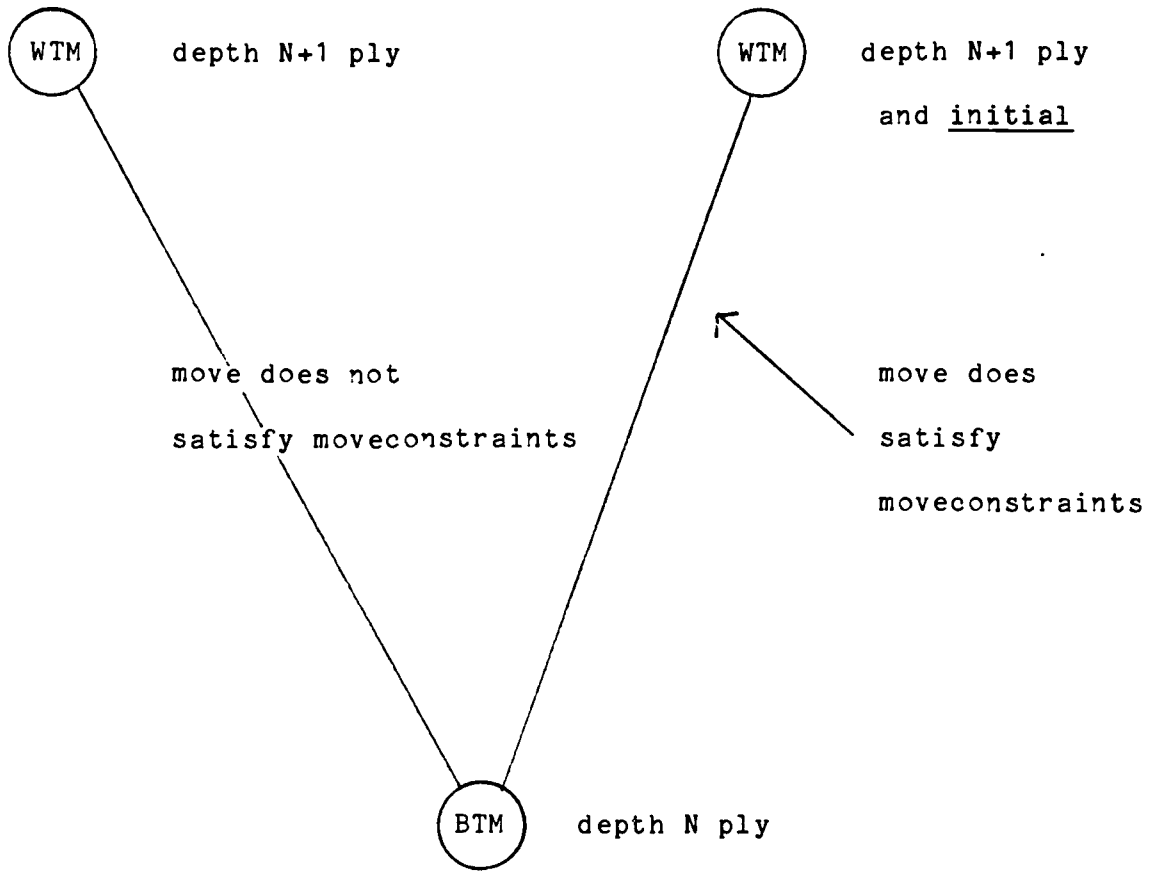
The first difference is in move constraints. These operate at ply 1 only in AL1 and specify which classes of moves are to be considered and in what order. For example the move constraint KMOVE in the KRK table means consider only king moves at ply 1 in

the forcing tree. Backing up for pieces of advice where move constraints are in operation proceeds as for the standard backup procedure except that when backing up to White to move (WTM) positions, those in which the backed up move satisfies the move constraints are marked as initial, that is they may be the root node of some forcing tree. Figure 4.2 illustrates the backing up process for move constraints.

The second difference is in the holding-goals of a piece of advice. Every node except the root node, of a forcing tree must satisfy the holding-goals. When backing up it is therefore necessary to check that no positions are backed up from which do not satisfy the appropriate holding-goals (the root node of the forcing tree in AL1 does not have to satisfy the holding goal which is why we have to back up to every position). In the case of KRK which has as holding-goal in BTM positions not stalemate we need only ensure that the holding-goal is satisfied at the terminal nodes since any position to which we can back up cannot be stalemate.

A third difference arises from the existence of what I shall call comparison predicates (CP's). These are predicates which necessitate a comparison between the root (or initial) nodes of a forcing tree and other nodes. An example from the KRK advice table

Figure 4.2



Backing up using move constraints

is the better-goal NROOMLT. This is true when the function room(position) has a smaller value at the terminal nodes of the forcing tree than at the initial node. The value of room(pos) is the number of squares to which the Black king is confined by the White rook. If the number is not well determined (when the rook does not confine the king) room(pos) takes value 100.

Since we are backing up from the terminal nodes of all possible forcing trees for a piece of advice, any evaluation of the better-goal NROOMLT has to be conditional - we don't yet know the value at possible root nodes. We are thus faced with two problems when backing up.

Firstly we have to minimax the value of the CP as we back up the tree. That is, the value must be maximised at us-to-move (UTM) nodes and minimised at them-to-move (TTM) nodes (given that the value of the CP is to be maximised). This ensures that US can find the optimal forcing tree from any position and THEY can find the best attempt at refutation. UTM positions are marked as initial iff the value backed up to that position is greater than the value of room evaluated in that position.

The second problem is that nodes may now appear more than once in the forcing tree. Consider a WTM position P backed up to at level d with value V. If it is backed up to at level d+2n with value V+v then we must back up from it again as we may now find

nodes that were not initial with room(P)=V but are with room(P)=V+v. This complicates the backing up process considerably since we must now take care that the BTM counter is not incremented backing up from P again. By chance this problem does not arise with the KRK advice table, as those pieces of advice with CP's are of depth 1.

Finally another change to the standard back up algorithm (given in Appendix 2) has been introduced to allow the production of the moves that would be played by a piece of advice. When an initial node is found during backing up the associated move is stored with the position. A full description of these modified backing up algorithms may also be found in Appendix 2.

The output of the backing up procedure for some piece of advice A is a list of all positions in the KRK domain that would satisfy the advice if given ab initio, plus the moves that would be played from each position. This in itself does not tell us which moves the advice table would play from any given position, since pieces of advice are applied in a fixed order of precedence. Therefore with each WTM position we must associate the moves that would be played by the highest ranked piece of advice producing a forcing tree in that position.

4.4 Formal definition of procedures

We have to make slight changes to the model of chapter 2 to accomodate forcing trees.

1. The depth of a forcing tree is limited. Therefore we only consider I_j : $j <$ maxdepth, in our solution.
2. We previously chose only BTM positions as terminal. In a forcing tree they can also be WTM.

4.4.1 Better goals only

The following is a solution for a piece of advice with *better goal* only. The better goal takes one argument.

$$I_0 = \{ v \mid bg(v) \wedge wtm(v) \}$$

$$I_1 = \{ v \mid (bg(v) \wedge btm(v)) \vee (v \in S^{-1}(I_0) \wedge S(v) \subseteq I_0) \}$$

$$I_{2i+1} = \{ v \mid v \in S^{-1}(I_{2i}) \wedge S(v) \subseteq \bigcup_{o}^{2i} I_j \} - \bigcup_{o}^{2i} I_j$$

$$I_{2i+2} = \{ v \mid v \in S^{-1}(I_{2i+1}) \} - \bigcup_{o}^{2i+1} I_j$$

we assume here that the maximum search depth is even, as it is in AL1.

4.4.2 Move constraints at ply one

To apply move constraints we define a set *initial*(V). If $v \in$

initial(V) then there is a forcing tree, satisfying the better goal and move constraints, rooted in v.

$$\text{initial} = \{v \mid v \in \bigcup_{0}^{\omega} I_j \;/\backslash\; \neg\exists v' \in S(v) \text{ s.t. } v' \in \bigcup_{0}^{\omega} I_j \;/\backslash\; mc(v,v')\}$$

4.4.3 Holding goals

We assume that holding goals are unary predicates.

$$I_0 = \{ v \mid bg(v) \;/\backslash\; wtm(v) \;/\backslash\; hg(v) \}$$

$$I_1 = \{v \mid (bg(v) /\backslash hg(v) /\backslash btm(v)) \;\backslash/\; (v \in S^{-1}(I_0) /\backslash S(v) \subseteq I_0 \}$$

$$I_{2i+1} = \{ v \mid v \in S^{-1}(I_{2i}) \;/\backslash\; S(v) \subseteq \bigcup_{0}^{2i} I_j \;/\backslash hg(v)\} - \bigcup_{0}^{2i} I_j$$

$$I_{2i+2} = \{ v \mid v \in S^{-1}(I_{2i+1}) \;/\backslash\; hg(v)\} - \bigcup_{0}^{2i+1} I_j$$

4.4.4 Comparison predicates

Let cp(v) be the value of position v, and assume the better goal is to increase the value of cp. We define a set of functions B satisfying,

$$B_0(v) = cp(v) \text{ for all WTM positions } v$$

$$B_{2i+1}(v) = \min(\{ B_j(v') \mid j < 2i+1 \;/\backslash\; v' \in S(v) \})$$

$$B_{2i+2}(v) = \max(\{ B_j(v') \mid j < 2i+2 \;/\backslash\; v' \in S(v) \})$$

$$\text{initial} = \{ v \mid \exists j\, (B_j(v) > cp(v)) \}$$

4.5 Proof Procedures

In this section we describe the two proof methods used to validate the KRK advice table.

The first is an automated version of the proof given by Bratko (Bratko 1978). His proof rests on 5 lemmas, which can be proved by hand but are tedious and of some length. The mechanical proof of the lemmas relieves the prover of the tedium and facilitates construction of correct lemmas, since lemmas can be conceived and checked quickly.

It is inevitable that for more complicated domains such as KRKN some form of 'automation' of the proof is essential to prevent the prover expending vast amounts of time finding and proving lemmas about various special cases.

The second proof method is completely automated, relying solely on the structure of the forcing trees used in the advice table to provide an exhaustive proof of correctness.

At this point it is appropriate to mention two problems that face any correctness proof of an advice table. The first is the

non-determinism of the system. Given a WTM position, the move played by the table will in general depend on the move generator used, and will not be uniquely defined by the formal characteristics of AL1, in the sense of those characteristics used as the basis of Bratko's correctness proof. This in itself is a natural and desirable consequence of the advice language structure and arises since there is in general no unique forcing tree satisfying a given piece of advice. It does however complicate the completely automated proof. An example of the complications that may arise is found by considering Bramer's exhaustive verification considered earlier. If his program were non-deterministic and produced 5 or 6 possible moves per position as does the KRK advice table it would take 5 or 6 times as long to verify the program.

Non-determinism is not desirable in itself. It is however desirable to specify the AL1 language at a level of abstraction above that of particular legal move generators, since the spirit behind using pieces of advice is that as long as a better goal can be achieved the particular manner in which this is done is irrelevant. If it is desirable to specify that particular moves are to be examined first this can be done via the move constraint mechanism.

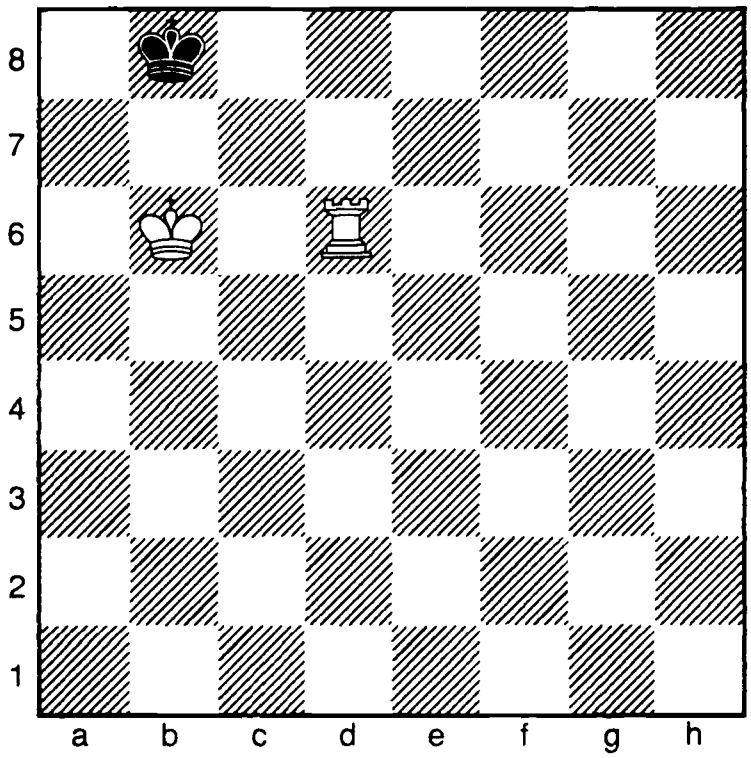
The second, more serious, problem arises from the two possible playing modes for AL1. In weak mode a new forcing tree is grown from each position and the relevant move played. In strong

mode however once a forcing tree has been found it is played through to the end. This is necessary to ensure that the better-goals are achieved. For example, if advice MATE in the KRK table had as better-goal mate, and a depth limit of 3 ply without the move constraint CHECK then ANYMOVE, the following situation could arise: Starting position Figure 4.3. The advice finds the forcing tree WRc6, BKa8, WRc8 mate and so plays WRc6. Black plays BKa8. Playing in weak mode the advice 'finds' WRd6, BKb8, WRd8 mate and so plays WRd6 whereupon BKb8 repeats positions. Black can claim a draw after 3 such repetitions.

With the KRK table we find that this form of loop is only possible for advice MATE or advice DIVIDE, the other three being of depth 1 ply when strong and weak mode are identical. Furthermore since MATE has as moveconstraint CHECKMOVE then ANYMOVE the above situation cannot occur, the table will always find a mate in 1 ply if it exists. This leaves us with advice DIVIDE. This can only be applied once from any starting position. It will not be applied again since its better-goals are holding goals for the other pieces of advice (except MATE). To prove the table correct we need only consider WTM positions where the better-goals of DIVIDE apply.

4.6 Mechanical proof of Bratko's lemmas

Figure 4.3



White to move

In 'weak' mode White can
get into an infinite loop via the move
sequence Rc6, Rd6 while attempting to satisfy
advice 'MATE'.

A major difficulty with correctness proofs, even for these simple endings hinges on the distinction between correct and optimal play. Following Bramer we define optimal play as that which improves the minimax-optimal number of the position; correct play as that which will always win, though the number of moves necessary may exceed the minimax-optimal number (ignoring the 50 move rule). Bratko proves correctness for a non-optimal program, without resorting to exhaustive enumeration of the moves played by the program.

Bratko's proof shows that there is a quantity room(pos) (the area in which the White Rook traps the Black King in position pos) which is always decreased by the table, although to do so may take several moves. When the value of room(pos) is 2, and certain other conditions hold, the Table can always force mate. The role played by room(pos) here is similar to that of minimax-optimal value in Bramer's proof of program optimality mentioned above, in that it is a numerical function over positions that can be shown to decrease.

The Advice structure enables a proof method reminiscent of Floyd's for program correctness. Given the 'pre-conditions' of positions satisfying a piece of Advice, we can ascertain the 'post-conditions' from a knowledge of the holding-goals and better-goals of the Advice. Floyd suggested a similar treatment

for program statements in programming languages. The aim there is to prove that if a certain set of pre-conditions are true before executing a statement then a related set of post-conditions will be true after execution of the statement. Bratko's proof of his five lemmas is given in (Bratko 1978). The lemmas are reproduced below and an alternative mechanical proof sketched for each. The relevant proofs were carried out, as described, on the Edinburgh/ICF DEC-10 machine.

LEMMA ONE

"For every legal White-to-move KRK position P, s(DIVIDE,P)"

METHOD:

, Initialise all legal Black-to-move positions P where RDIVIDES(P) holds. Back up 5 ply. Check that every legal White-to-move position is marked at depth 1,3 or 5. This shows that DIVIDE is satisfiable within 5 ply from any legal White-to-move position.

LEMMA TWO

"In every Black-to-move position P that satisfies [RDIVIDES(P) and ~REXPOSED(P)] or LPATT(P) any legal Black move results in a position Q satisfying [RDIVIDES(Q) and ~REXPOSED(Q)] or LPATT'(Q)"

METHOD:

It is sufficient to prove that LPATT(P)=>LPATT'(Q) and [RDIVIDES(P) and ~REXPOSED(P)] => [RDIVIDES(Q) and ~REXPOSED(Q)]. The lemma was proved in this way as Bratko's proof seems to indicate that LPATT'(Q) is defined as being true in any possible successor position of LPATT(P). The lemma then reduces to the second clause. Initialise all White-to-move positions satisfying the RHS of the above implication. Backup 1 ply. Show that all the marked Black-to-move positions satisfy the LHS of the above implication.

LEMMA FOUR

"For every White-to-move position P, RDIVIDES(P) and ~REXPOSED(P) and MDIST(OK,CS,P)>2 and ROOM(P)>2 => s(APPROACH,P)".

METHOD:

Find all positions P where s(APPROACH,P). Let this set be S'. Let S={set of positions satisfying LHS of implication above}. Check that $S \subseteq S'$.

LEMMA FIVE

"Starting to play from any White-to-move position P satisfying [RDIVIDES(P) and ~REXPOSED(P) or LPATT'(P)] and

MDIST(OK,CS,P)$\leq$2 and ROOM(P)>2 the KRK table forces squeezing in at most three moves"

METHOD:

The method is to generate S={all positions from which squeezing can be forced in 3 moves}.

1. Let S'''={all White-to-move positions from which squeezing can take place}
2. From S''' generate all Black-to-move positions that lead to elements of S''' (by backing up 1-ply). Call this set Q. From Q generate all positions from which the table might play, let this be S''.
3. applying this procedure again get S' from S''. Then S=S'''$\cup$S''$\cup$S' is the required set. It remains only to check that the set defined in the lemma is contained in S.

LEMMA THREE

"Starting from any legal White-to-move KRK position P that satisfies ROOM(P)=2 and ~REXPOSED(P) the KRK table always mates."

METHOD:

1. Show the set S defined by the conditions of the lemma can be decomposed into disjoint subsets M, A, K s.t. M$\cup$K$\cup$A=S where m$\in$M => s(MATE,m), a$\in$A => s(APPROACH,a), k$\in$K => s(KEEPROOM,k).
2. Backup 3 ply from positions with Black-to-move in checkmate, show that this gives M.
3. Produce set M' from M by backing up 1 ply and then finding the White-to -move predecessors under APPROACH

or KEEPROOM. Iterate this procedure and show that for some n, $S \subseteq M^{n}$.

As can be seen the proofs of these lemmas are conceptually fairly simple and furthermore a great deal of case checking has been eliminated.

4.7 Mechanical Proof

This second proof method relies on the fact that the AL1 system specifies (within the limits mentioned above) the move to be played from any given position. With this in mind we can find a simpler proof than that given above, albeit one less transparent to the human reader.

We have mentioned previously the difficulty of proving the correctness of non-optimal strategies. Clearly attempting to use the minimax optimal values generated by a KRK database would be fruitless here, so we use another complementary approach; that of considering the Advice Table strategy as a directed graph in the KRK position space.

The nodes of this graph are all the legal White-to-move positions plus all those Black-to-move positions to which the table plays. Ignoring complications we have one exiting arrow

from every White-to-move node and as many, from every Black-to-move node as there are legal moves. It is clear that if the program does not play correctly there must be a cycle in this digraph. That is there must be some White-to-move position P from which there is a path back to itself.

The problem reduces then to finding whether the digraph has any cycles. This is made simple by the reduction we have made from a 'game' graph to one where there is only one player with choice (ie. Black). Algorithm 1 below finds any positions from which the advice table may not win quickly. Note that there is a limit to the amount of work the algorithm has to do since it never has to find successors for more than 30,000 Black-to-move positions (the whole position space). This algorithm is similar to one given by Bramer (Bramer 1980a). His algorithm is less general since it can determine that the graph is cycle-free only if White has one possible move from each position. As we have seen the AL1 system is non-deterministic and so several moves are possible in any position. Algorithm 1 can accept several possible moves from any White-to-move position. Algorithm 2 is a more direct, less computationally efficient approach. It has the advantage that it can find a particular cycle in the graph, rather than just positions from which cycles are possible. This could be helpful in debugging an advice table. Also, if an advice table produces incorrect play this algorithm can terminate with a counterexample faster than Algorithm 1.

ALGORITHM ONE

1. Start. Initialise all positions with Black-to-move in checkmate. Set these as lost depth 0. For every Black-to-move position in the position space set a counter with the number of legal moves for Black from that position. For every White-to-move position set a counter with the number of different moves the advice table may play in that position. Set i = 0.

2. Back up from positions lost with BTM by generating for each BTM position lost at depth i the set of predecessor (WTM) positions, from which the advice table may play. Mark each of these predecessors as won for White depth i+1 if not previously marked as won and if the (WTM) counter is 1, otherwise decrement the (WTM) counter by 1. Set i to i+1.

3. Back up from positions won with WTM by generating, for each WTM position won at depth i the set of predecessor positions (all the legal Black moves to the WTM position). Mark each of these won for White at depth i+1 if not previously marked as won and if the (BTM) counter is 1, otherwise decrement the (BTM) counter by 1. Set i to i+1.

4. When there are no more positions left to back up by 2 or 3, check if there are any White-to-move positions left unmarked at some depth. If there are, then there are cycles in the graph and the advice table may not win from unmarked positions.

NOTE

The depth at which the backup procedure terminates gives the maximum number of moves the Advice Table requires to mate. Since every White-to-move node is marked with a depth (since the strategy is correct!) a statistical investigation of the strategy is possible by comparing these depths with the minimax-optimal ones.

ALGORITHM TWO

1. Start. Mark all Black-to-move checkmate positions as terminal.

2. Pick any White-to-move position, not terminal, as root node.

3. Search, depth first, from this node until terminal positions are reached. From White-to-move nodes only search those moves the Advice Table plays. During the depth first search a node in the search tree is normally reached twice; once going down, once coming up. The first time a node is reached mark it <u>used</u>. If the node occurs again before the search returns through it we have a loop, finish, else when the node is reached a second time delete <u>used</u> and mark it terminal.

4. When the search terminates, if there are any White-to-move nodes not marked terminal, goto 2 else finish

NOTE

This algorithm depends on the fact that if during the search one reaches a node marked terminal there are no cycles through that node. Again the amount of work the algorithm has to do is strictly limited.

These algorithms cope easily with one of the two problems I mentioned earlier, that of non-determinism. If more than one move may be played from any White-to -move position, all are included. The second is more intractable.. Algorithms 1 and 2 can prove only that an Advice Table always mates when playing in weak mode. The two modes are equivalent for KRK since the maximum depth of search is 1 for the relevant pieces of advice, but this is not so in general.

Algorithm One was implemented in BCPL on the local DEC-10. The backing up procedure was fast, using less than 15 minutes CPU time. As expected the KRK table proved correct. Its correctness was determined using algorithm 1 above. This backing up excluded DIVIDE as explained earlier. Table 1 below gives the number of positions won at various depths (in ply) using the KRK table. Since DIVIDE always succeeds in less than or equal to 5 ply we can put an upper limit to the length of play of 69 ply, somewhat less than Bratko's upper bound of 77 ply. The number is 69 here rather than 68 since Black has a move after White has succeeded with DIVIDE.

CONCLUSION

We have demonstrated a method of exhaustively finding the moves found by a piece of advice from every position in the space of an endgame. This technique has been applied to the endgame of King and Rook vs. King in two ways. A series of lemmas was verified, and an exhaustive proof was given.

This technique suffers from two limitations. The overriding limitation is that the size of the position space must not be too large to fit onto the primary or secondary storage of a computer. This effectively limits the size of the space to the order of one hundred million positions with current technology. The second limitation is that solutions cannot be found in a reasonable time for every possible specification of a piece of advice. The

restrictions are that holding goals should be unary predicates on positions, or should take on only a small range of values and that move constraints should apply only at ply 1 in the forcing tree.

In addition we have noted that if play is in strong mode, with depth of search greater than 2 ply, the method of exhaustive verification is inapplicable and the proof must proceed by the verification of lemmas. Similarly if no better goals are specified for pieces of advice then it is usually necessary to proceed by exhaustive verification rather than the proof of lemmas.

TABLE 4.1

Depth in ply	Number of positions
1	189
3	587
5	16
7	10
9	33
11	34
13	73
15	87
17	139
19	117
21	187
23	246
25	416
27	385
29	536
31	829
33	772
35	880
37	1285
39	1214
41	1027
43	1413
45	1394
47	1064
49	1151
51	1215
53	830
55	481
57	344
59	142
61	142
63	31

There are 4144 positions satisfying DIVIDE

5. The King and Rook vs. King and Knight Ending

5.1 Bratko's original advice table

Bratko's first advice table for the KNKR ending is shown in Figure 5.1a.

The predicate descriptions, and a description of the pieces of advice involved, can be found in appendix 4. Any KNKR position falls into one of four classes determined by the four rules in the table. Rule CR deals with the exceptional case of Figure 5.2 where the only drawing move is a counterintuitive separating move of the knight. This is the only case of such a move described in the textbooks on the ending. Rule R1 deals with positions where the king and knight are not separated and the king is not on the edge of the board. This class of positions is usually drawn, the textbook suggests that the king and the knight remain together and that the king should strive to remain in the center of the board. Rule R2 deals with positions where the king and knight are close but where the king is on the edge of the board. In this case the textbook suggests that the two pieces should remain close and the king should stay away from the corner of the board. The final rule

Figure 5.1a

	RULES			
Predicates	CR	R1	R2	ER
OKEDGE	-	N	Y	-
OKONSEP	-	N	N	-
CORNCASE	Y	-	-	-
Advice lists	9	1	1	1
		2	5	10
		3	6	11
		4	7	12
		11	8	
			11	
			12	

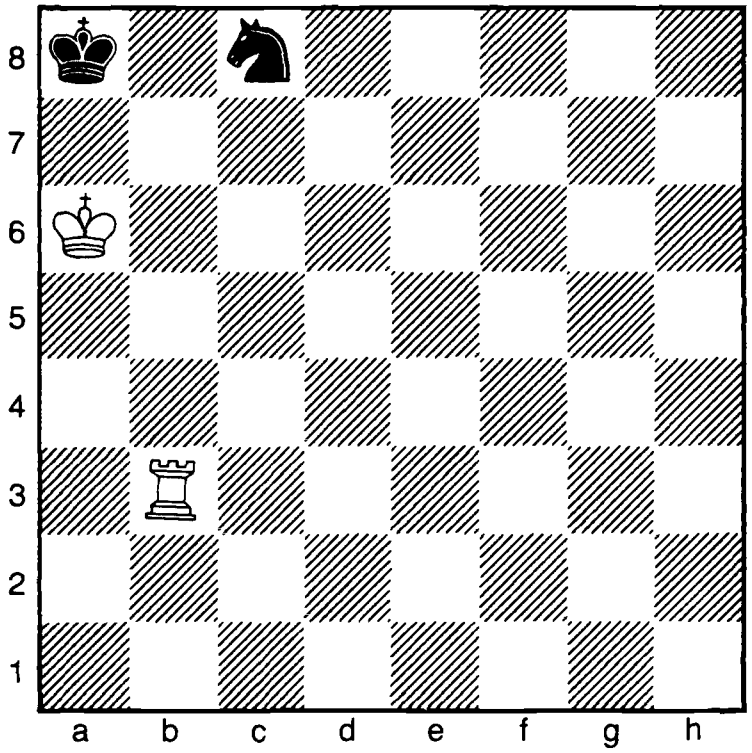
Bratko's original Advice Table for KNKR

Figure 5.1b

	RULES				
Predicates	CR	RO	R1	R2	ER
OKEDGE	-	-	N	Y	-
OKONSEP	-	-	N	N	-
CORNCASE	Y	-	-	-	-
EDGECASE	-	Y	-	-	-
Advice lists	9	16	1	1	1
			2	5	15
			3	6	10
			4	7	13
			11	8	14
				12	11
					12

Bratko's revised Advice Table for KNKR

Figure 5.2



Black to move

Black's only drawing move is ...Ne7.

CORNCASE

ER deals with positions where the king and knight are separated. It was thought that these positions were almost always lost, since the enemy king and rook could cooperate to isolate and capture the knight. Bratko discovered, while programming the ending a further heuristic of not moving the king and knight a knight's jump from each other. This heuristic is extremely powerful in many positions, and is not given in any textbook. The intensive work Bratko did in formalising rules of play forced him to make explicit heuristics such as this, which chessmasters have not documented.

The strategy Bratko used in the table is a gradual weakening of the holding goals as we progress down the advice list for each rule. This ensures that we choose the move which satisfies the most stringent possible set of holding goals.

The form of this table demonstrates the differences between AL1 and Huberman's model. In the KNKR table the notion of stage has to be abandoned, since the goal is to preserve the draw rather than to make progress in any way. The emphasis is almost entirely on holding goals, which Huberman used only to restrict the depth of search. The AL1 strategy for the ending might be described as; to maintain the draw stay out of trouble for as long as possible. Most of the pieces of the advice in the table are satisfiable if and only if the appropriate holding goals can be maintained for 4 ply. It is worth noting that this strategy is <u>not</u> possible to

implement in any way using a conventional minimax search, since such a search is concerned only with the value of positions at the leaves of the search tree. The pieces of advice in the AL1 strategy consider the value of the holding goals at all leaves in the tree, and since there is no better goal cannot be considered as merely pruning the tree.

The automated proof techniques used to solve the KRK ending are also suitable to provide a proof of the correctness of the KNKR strategy. The proof process is in essence simpler for this endgame for two reasons,

1. The table plays in weak mode, using only the first move of any forcing tree generated.

2. We are seeking only to preserve the draw, rather than win. There is therefore no need to check for circuits in the game-graph. A database for the ending can be used directly to determine whether a position is won.

The proof procedure entails creating, for each of the twelve pieces of advice a lookup table listing the positions satisfying that advice, and whether the move played maintains the draw (for those positions that are drawn). Holding goals which use 'comparison predicates' can be treated by considering each possible value in turn for possible <u>initial</u> positions. This is feasible as there are only two such predicates, both of which take values in the range 0 - 3. These are OKEDNDGE and OKCDNGE. The major drawback with this proof procedure is its computational expense, roughly an order of magnitude greater than that of

creating a database for the ending. There does not seem to be an alternative, it is unclear how a more formal proof could be constructed without any ability to characterise drawn positions in a static manner.

Although this procedure is computationally expensive, it is far cheaper than running the AL1 system on every position in the space. At an average of 1 CPU minute per position this would take approximately 25,000 CPU hours with the current implementation of AL1. On the positive side, once we have computed the lookup tables, we do not need to use the AL1 system in computing moves - unless its explanation capability is required. If we assume 40 CPU hours on a DEC 10 (KL processor) for creation of the lookup tables the break even point is 2400 position evaluations, again allowing 1 minute of CPU time per position evaluation for the AL1 system.

5.2 Errors in the advice table

Use of the KRKN database revealed several positions where the Black king and knight are separated and where only counterintuitive moves, resulting in further separation, preserve the draw. Using the database to verify their game-theoretic value, Senior Master Danny Kopec compiled a benchmark of twenty positions, all falling under rule ER in the table, where the table

erred. An example of such a position is shown in Figure 5.3. Although the king and knight are separated Black can hold the draw with Nf6. The table plays Nc3, an 'intuitively' correct move, bringing the two pieces together. This move loses to a long refutation.

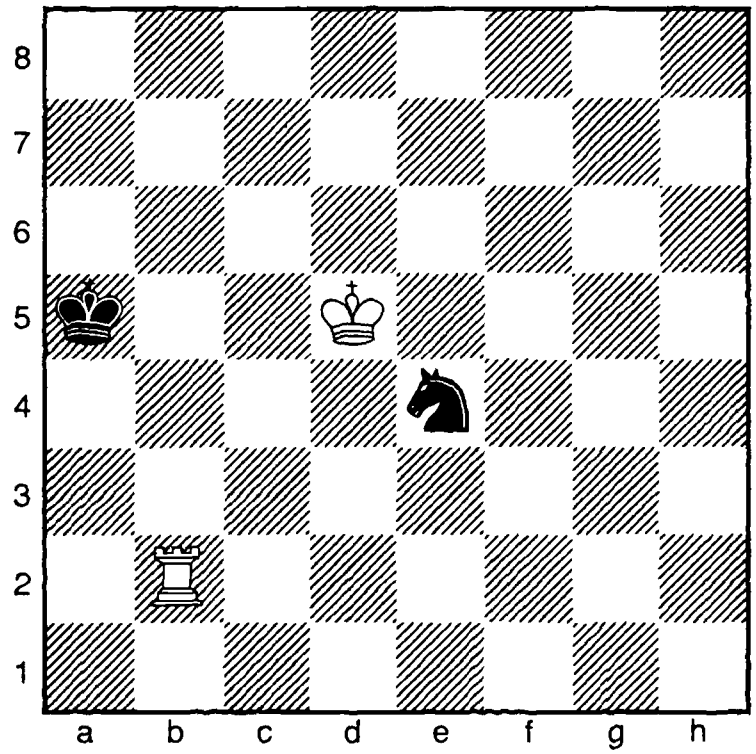
The existence of this class of chess positions is largely undocumented by chess theory. This is one of the first cases where computer analysis of chess has shown large gaps in the findings of chess theory, and provides an opportunity for an analysis of exactly how and why this happens.

5.3 An analysis of a textbook description of KNKR

The above results prompted a detailed investigation of the analysis of KNKR given in chess textbooks. The most detailed sources in English are Fine (1941) and Averbakh & Maizelis (1974). Fine's analysis is based on that of Berger, whereas Averbakh relies on many more sources. Fine's account was chosen since his analysis is more detailed, and he was the source Bratko used in programming the KNKR ending.

To show the opacity of the ending even in the face of extensive analysis consider the position of Figure 5.4. This

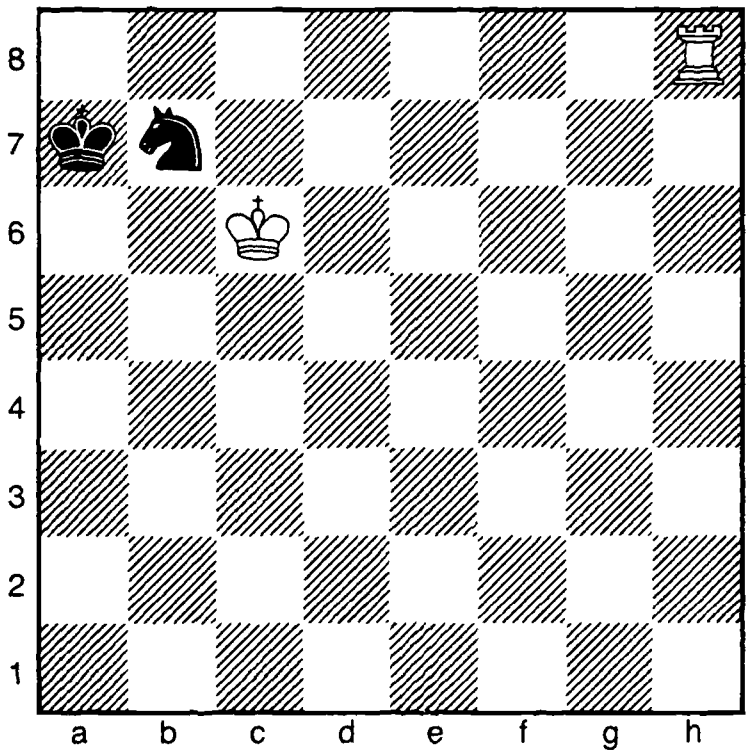
Figure 5.3



Black to move

...Nf6 draws, all other moves, including the plausible ...Nc3, played by the original KNKR table, lose.

Figure 5.4



Black to move

A variant of the 9th century al-Adli position

position is a variant of that given for the ancient game of Chaturanga by al-Adli in the ninth century. The king, rook and knight are the only three pieces to have the same moves in the two games, which makes this one of the oldest positions known to chess theory. It was rediscovered in 1859 and (incorrectly) analysed in 'The Chess-Players Chronicle'. Berger subjected the position to extensive analysis (also incorrect), and this was used by Fine. The KRKN database shows that the position is won in 14 moves.

To show the difficulty of analysis I present Fine's main line for the first few moves, comparing his move with the minimax-optimal move given by the KRKN database.

1	...	Na5 +	forced
2	Kb5	Nb7	optimal
3	Rf8!	...	(Fine's punctuation)

Rh5 is best, 1 move shorter. Here Fine's notes say 'Not ...Rh7, because of Kb8,Kb6 drawn'. However Kc6 wins, and more quickly than Fine's main line. This is one of two cases where Fine gives the wrong game-theoretic value for a position.

3	...	Nd6 +	forced
4	Kc6	Nc4	
5	Rd8!	...	

In spite of Fine's exclamation mark, this move changes the

depth of win from 11 to 17 moves. This exemplifies a common phenomenon. The chessmaster finds a plan which preserves the game theoretic value of the position, but which is inefficient in the minimax-optimal sense. The move Rd8 is well motivated, in that it follows a plan to separate the king and knight, yet Rf4 is more efficient even though it requires a more thorough analysis.

All the main lines given by Fine have been analysed and Figure 5.5 illustrates the results. It is surprising that there are so few non-optimal moves at large depths and so many at relatively shallow depths. However, if the win is very deep there is usually only one good move. At depths 7 - 10 the master doesn't try to find optimal moves, just good ones. At a lower depth still he can see through to the end.

Using the database in this way we see that the ending is extremely hard to analyse. There is a small subset of the position space, the longest wins, which has never been explored and was previously unknown to chess theory. Masters have great difficulty in winning from these positions. In our small scale trials of human play against the database we never saw it done. Subsequently however similar tests were passed by A.J. Roycroft (Roycroft 1980), a specialist in endgame play who had studied some of the very deep wins.

Figure 5.5a

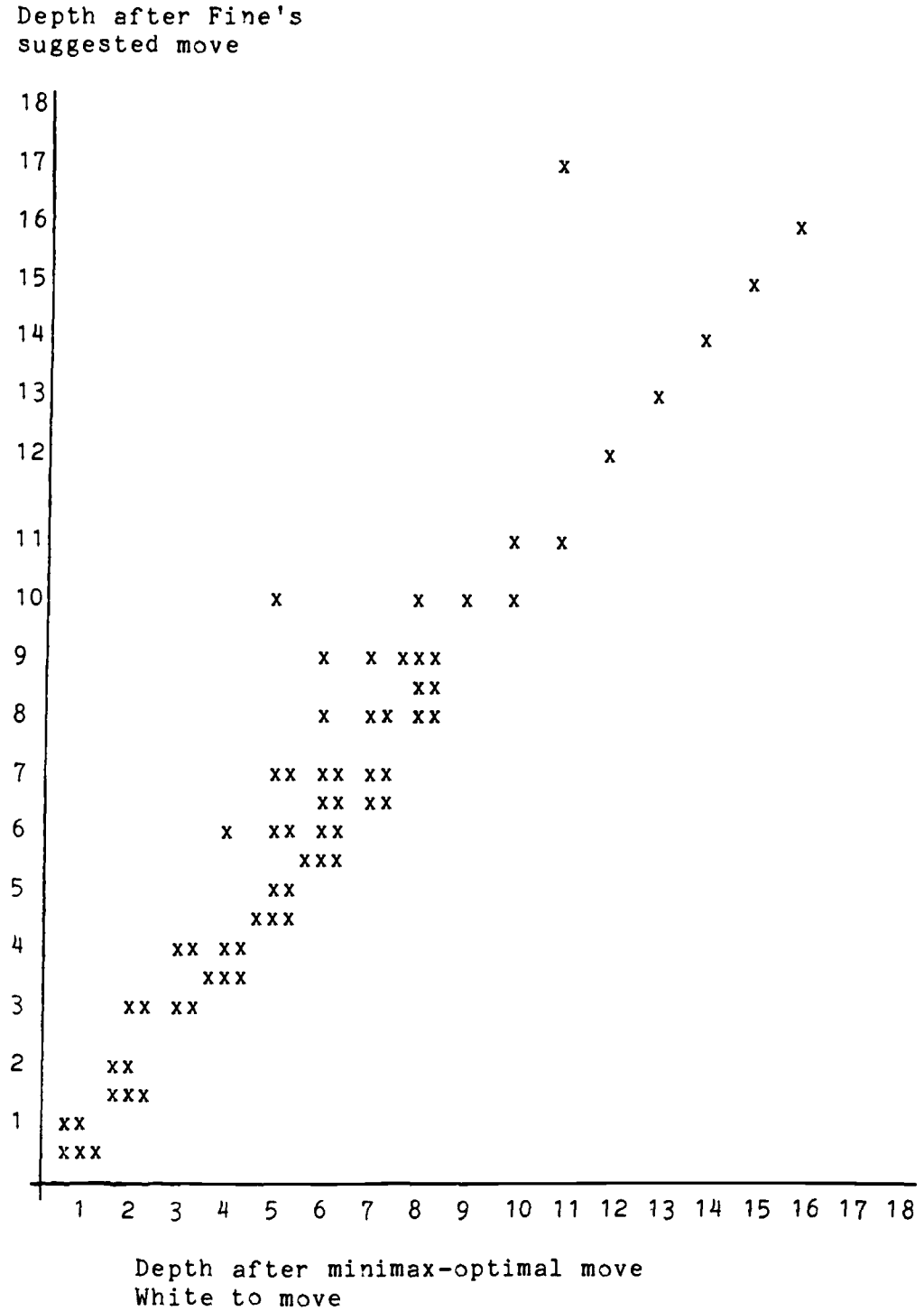
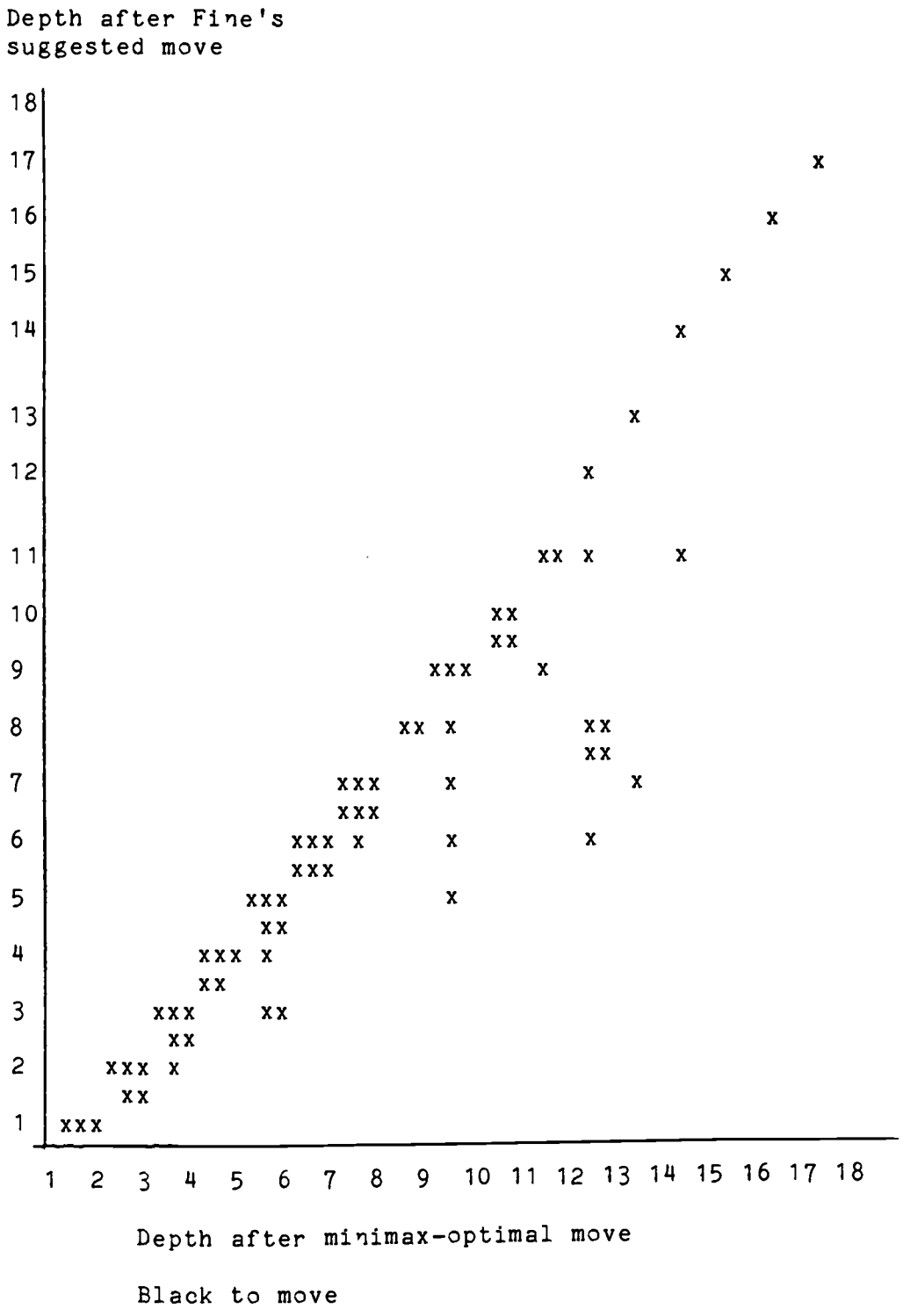

Depth after Fine's
suggested move
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Depth after minimax-optimal move
White to move

Figure 5.5b


Depth after Fine's suggested move
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
Depth after minimax-optimal move
Black to move

5.4 Correction of the KNKR table

As a result of the discovery of the class of positions drawn despite separation of the king and knight, Bratko revised his KNKR strategy. The intuition he used was that in these separated positions the knight's side should choose moves which give the knight a clear path back to the king. Most of the positions in Kopec's benchmark can be understood with this idea of path. The new version of the table incorporated this idea and solved all the positions in the benchmark. Bratko introduced one new rule, EDGECASE, to cope with the position of Figure 5.3 and changed rule ER. The position of Figure 5.3 requires a rule to itself for two reasons. Firstly, the pattern is not generalisable. If the position is moved down one rank it is lost, up one rank and the knight move near the king now draws. Secondly, the depth at which the loss becomes apparent considerably exceeds the 6 ply search used in the new version of rule ER.

Bratko introduced a strategic measure on positions, NSAFETY, to encapsulate the notion of path. NSAFETY is used as a move constraint. NSAFETY is a function of the positions of the White king, Black king and Black knight, the three pieces whose positions vary relatively slowly and so provide the basis for a positional measure. Moves with a high value of NSAFETY are selected to be examined first, using the move constraint facility of AL1. Since move constraints may only be applied at ply 1 the

measure contains an implicit lookahead. Another new feature is that the moves at ply 1 are totally ordered by the move constraints. This had not been done before in an advice table.

The NSAFETY measure uses a compound criterion to assess the value of moves,

1. Try to minimise the distance between the Black king and the Black knight
2. Try to maximise the distance between the White king and the Black knight
3. Prefer positions in which there are more knight's paths back to the Black king

In addition to this strategic measure some extra tactical features were included:

1. A 6 ply search with immediate loss of the knight detected statically
2. Beyond 4 ply it is necessary to consider only those opponent moves which are checking moves, attacks on the knight or moves to constrain the knight's mobility

This approach completes a revolution in the use of a piece of advice. Better goals are explicitly disavowed, because there are no clear goals to pursue within a 6 ply lookahead. Instead we have a search strategy which looks more like minimax search. A good deal of the knowledge in a piece of advice is invested in the move ordering - that is in the search strategy. One consequence of this is that the proof procedure described in chapter 4 is impossible

to apply. The implications of this extra complexity for the suitability of AL1 as a programming tool for the ending are considered in section 5.6 after discussing the performance of the revised advice table. In this section we see that AL1 is not in fact suitable for programming the KNKR ending, if we desire the resulting advice table to be correct.

5.5 The performance of the new KNKR table

At this stage 2 questions arise about the performance of the table.

- Does it perform correctly in separated positions?
- Does it perform correctly in positions with the king and knight together?

From our previous experience we were fairly sure that the new version of the table would fail in some separated positions. It was believed however, that the table played correctly in non-separated positions. Extensive testing against a chessmaster had failed to reveal any flaws (Bratko, Kopec and Michie 1979) in its performance in this subdomain.

5.5.1 Separated positions

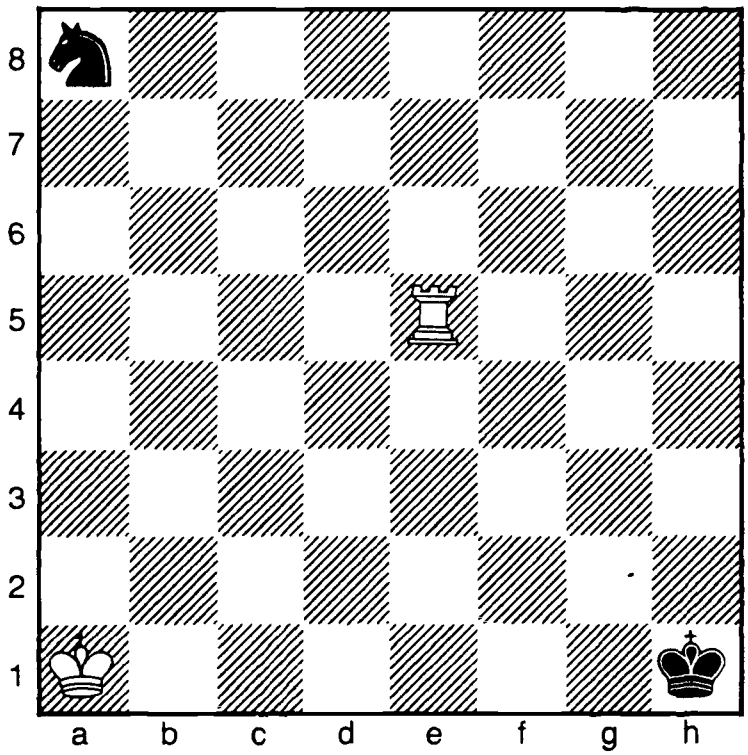
It is not feasible to use the automatic techniques of chapter 4 to verify play in separated positions with the new table. The NSAFETY measure is used as a holding goal and takes too many discrete values (over 256). It is easier to generate positions where the table is liable to err. Using the KRKN database positions were generated satisfying the following conditions:

1. Black-to-move (the knight's side).
2. The Black king and knight are separated (the king distance between them is greater than 3).
3. Black has only one drawing move (this can be discovered from the database without search).
4. Black cannot win the White rook immediately.
5. At least 1 Black move leads to mate or loss of the knight in more than 30 ply. This move leads to the successor position with the highest value of NSAFETY.

Positions passing through this filter were likely to be incorrectly handled by the advice table, since for the deep wins the knight and the king can remain together for many ply before being separated, nullifying the discriminating effect of the NSAFETY measure.

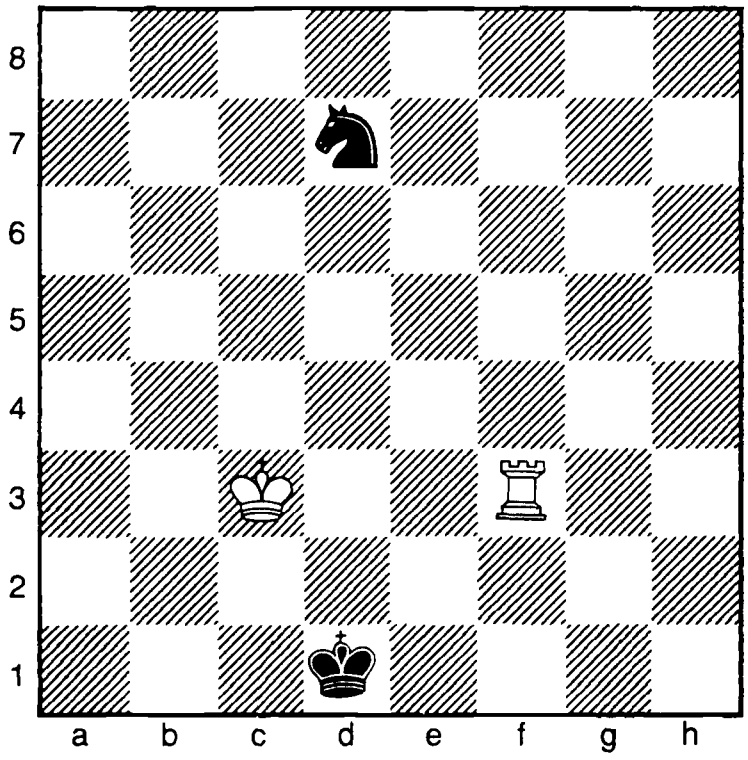
Over a dozen positions were found where the modified table plays incorrectly in separated positions, This is by no means an exhaustive list, less than 10% of the KRKN position space was examined, and the filter used is extremely restrictive. Some of these are shown in Figure 5.6. It is interesting to see why the table fails in these positions. In Figure 5.6a the correct move

Figure 5.6a



The advice table plays ... Nc7, correct is ... Nb6. The critical factor in determining the correct move is the position of the White rook - which is not considered by Bratko's NSAFETY measure.

Figure 5.6b



The advice table plays ... Ne5, correct is Ke2.

depends on the position of the White rook. If it were on d5 then Nc7 draws. The NSAFETY measure does not take the position of the rook into account and so cannot distinguish these positions. In Figure 5.6b the table fails to find the correct move because the tactical refutation of Ne5 is outside its lookahead horizon.

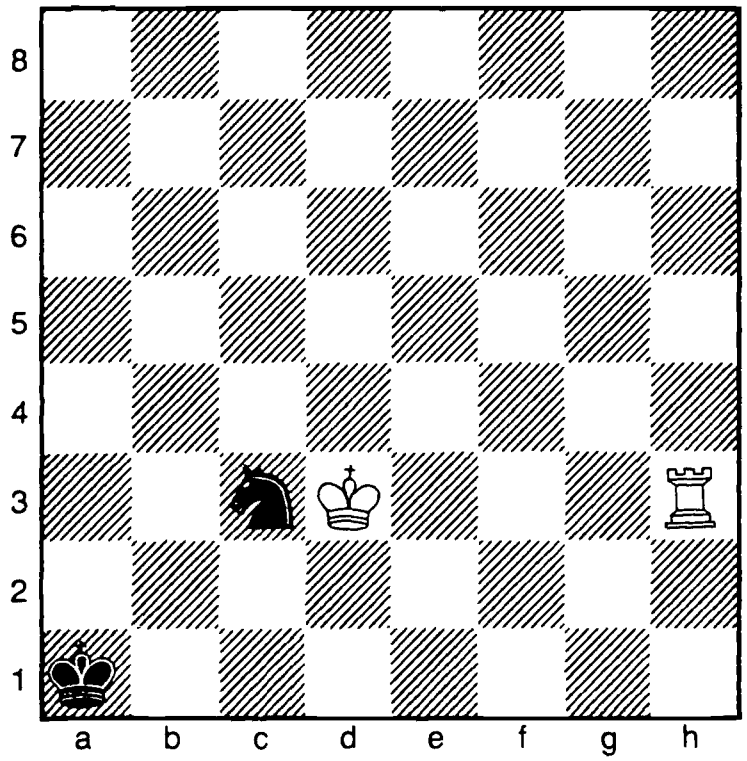
5.5.2 King and knight together

The experimental procedure was changed slightly. The tests were made to discover errors in R2 which matches the pattern OKEDGE and not(OKONSEP). The intention was to find drawn positions from which the first piece of advice in the list for R2, HOLDEDGE1, is satisfiable and the move played by the table leads to a loss. White-to-move positions were generated satisfying:

1. Lost in more than 30 ply
2. The Black knight does not attack the Black king
3. TKONDGT1 (the White king and Black knight are more than 1 kingmove apart)
4. OKONDLE2 (the Black king and Black knight are less than 3 kingmoves apart)

These positions were backed up to drawn Black-to-move positions in which the Black king was no further from the edge or corner, thus satisfying the move constraints of advice HOLDEDG1, the first piece of advice in the list of rule R2. Two positions were found in which the table errs. These are shown in Figure 5.7. In Figure 5.7a, 1. ... Nb1 loses to 2. Rh2 Na3, 3. Kc3 Nb5, 4. Kc4. Figure 5.7b is of considerable appeal and interest. After 1.

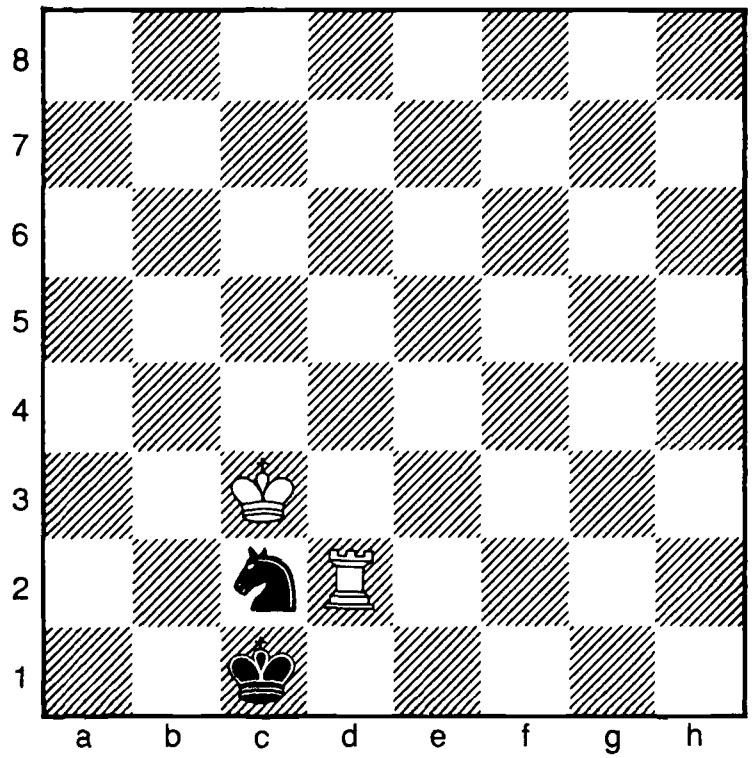
Figure 5.7a



Black to move

The table plays ... Nb1, correct is Kb2

Figure 5.7b



Black to move

The table plays ... Ne3, correct is Ne1.
...Ne3 loses to a long variation
(given in the text) which transposes
to a variation of the al-Adli position.

Ne3 the main line is 2. Kd3 Nd1, 3. Rc2+ Kb1, 4. Kd2 Nb2, 5. Rc1+ Ka2, 6. Kc3 Na4+, 7. Kb4 Nb2. This position is worthy of a mention, since it is a modification of the al-Adli position we saw earlier. The advice table fails because it is forced to play into one of the most analysed of KNKR positions, at a depth of 12 ply, well outside its lookahead of 4 ply.

5.6 Defects of AL1

Despite its failure in the positions described above, we must recognise that the AL1 system performs extremely well in this ending. Tests done in Edinburgh (Bratko 1979) indicate that it performs at approximately the same level as a human chessmaster.

The positions that have been found where the table errs require either a major modification of the table or the introduction of extra rules aimed solely at coping with these particular positions. The first approach would take too long, with no guarantee of the final result being correct. The second approach would produce an extremely large and unwieldy table, also with no guarantee of being correct. This is unattractive as we want the advice table and its output to be easily understood by humans. We claim the KRKN table, given in Fig 5.1a and described more fully in appendix 4, to be easy to understand, though this is

a subjective judgement.

The problem of programming exceptions has been noted before by Zuidema (1974), and can be seen in Bramer's work on KPK where modification of his program to produce optimal play doubled the number of rules. It is interesting to ask why coping with exceptions is harder with KNKR than with KRK. The strategy Bratko drew on for KRK never needs a deep search to achieve its goals. It is possible to decompose the strategy into goals which are always achievable within 5 ply, and which are easy to describe statically. This is the crucial point. In theory we can produce a static description of anything we like in the KNKR domain because it is finite. In practice we cannot easily and naturally produce descriptors to act as goals for a 4 or 6 ply search.

The advice table writer is faced with a dilemma. To cope with the complexity of the ending he has to use very complicated pieces of advice, but this makes it much harder to ensure correctness of the results. One possible way out is to create a more powerful system by upgrading the type of forcing tree used. With a more powerful tree searcher the structure of the individual pieces of advice can be simpler. Several suggestions have been made on these lines lines, and one system has been implemented.

The AL1.5 system (Mozetic 80) has been implemented and includes the following extensions to AL1,

1. Goal predicates can be defined by satisfiability of a piece of advice, which in effect facilitates recursive application of advice;

2. Move constraints can be specified for both sides;

3. Move constraints can define an ordering by assigning 'move values' to selected moves;

4. 'One side' game tree analysis is possible via a 'dummy move' facility.

Bratko and Michie (Bratko and Michie 1978) have specified the design of an AL2 system with the following extra features,

1. Move constraints can be applied at every level of the tree and by both sides, the move constraints can be ordered by a numerical measure;

2. Functions can be minimaxed with respect to better goals, this enables the selection of a 'best' forcing tree from a position;

3. Global better goals and holding goals can be applied which must be satisfied by every piece of advice that is called;

4. Forcing trees derived from high priority pieces of advice can be applied during execution of forcing trees from low priority pieces of advice. For example, if checkmate can be given this might override a forcing tree designed to win material.

Improvements such as these would certainly increase the power of the AL1 system. Unfortunately they do not increase our ability to prove the correctness of advice tables. This for two reasons. Firstly, in endings such as KRKN, search needs to be very deep in many variations before an appropriate 'better goal' is reached. This necessitates the use of complicated search constraints, such as the NSAFETY measure. These defeat the use of the better goals

as a means of defining the behavior of the program. We cannot be sure that if a piece of advice is not satisfiable, the better goal is unachievable. Secondly, the complex structure of the forcing trees makes direct mechanical proof extremely hard, as we have seen for Bratko's modified KNKR table.

There are other drawbacks with the AL1 approach for complex endings such as KRKN. Chess theory treats the play of the weaker and the stronger sides in KRKN in the same way. Exactly the same techniques of analysis are used for both sides. The Al1 strategy for KNKR cannot be used to create a playing strategy for the stronger side. The greatest drawback is the impossibility of training the AL1 system with examples. The ability to inductively acquire knowledge is paramount in complex endings, simply because of the volume of knowledge which the domain expert has to transfer. All this is not to suggest that it is impossible to produce a correct strategy for KRKN in AL1 using the automatic proof method given in chapter 4. To use the method however we must insist on a certain structure for the pieces of advice we use. A sufficient set of conditions is,

- Use of move constraints at ply 1 only
- Holding goals which are predicates on the current position only, or which do not involve the use of functions with large codomains, such as NSAFETY.

With pieces of advice in this form an iterative approach to a correct strategy is possible. The objection is that the strategy

found will, of necessity, be unwieldy.

Before suggesting characteristics of a system capable of correct play in the KRKN ending in a humanly understandable fashion, the al-Adli position is examined, to see what sort of reasoning is required in complex KRKN positions. It is important to realise that the solution of this problem is extremely difficult. It took human chessmasters almost seventy years, from 1859 to Frinck's 1928 solution, to find the optimal strategy for White, and this is by no means the most difficult of KRKN positions.

Before we begin we need to define the notion of a path between the Black king and knight which captures our notion of separation.

> Def A path between the Black king and knight is a sequence of squares $\langle s_0, \ldots, s_n \rangle$ such that for each s_i, $i < n$ one of the pieces can move to s_{i+1} without being attacked by more White pieces than Black and without the Black king being exposed to check.

In general if the minimum path length between the king and the knight is 4 or greater the two pieces are separated, and if other factors are favorable White can win. There are many exceptions to this heuristic, but it is adequate for our purposes.

There are two distinguishable phases of play in the al-Adli position.

1. The knight is driven into a separated position, using threats of mate against the king, pins, forks and skewers against the knight. The Black king can be kept in the corner to aid the mate threats.

2. The White king and rook are manoeuvered to trap the knight, whence it can be captured. The Black king has to be kept at a proper distance.

Starting from Figure 5.8 initial analysis reveals that:

1. The Black knight has limited mobility on b7, if the rook were on the d-file it would be paralysed

2. Black, to move, has only Nd6 available

3. The White king is in a position to keep the Black king in the corner

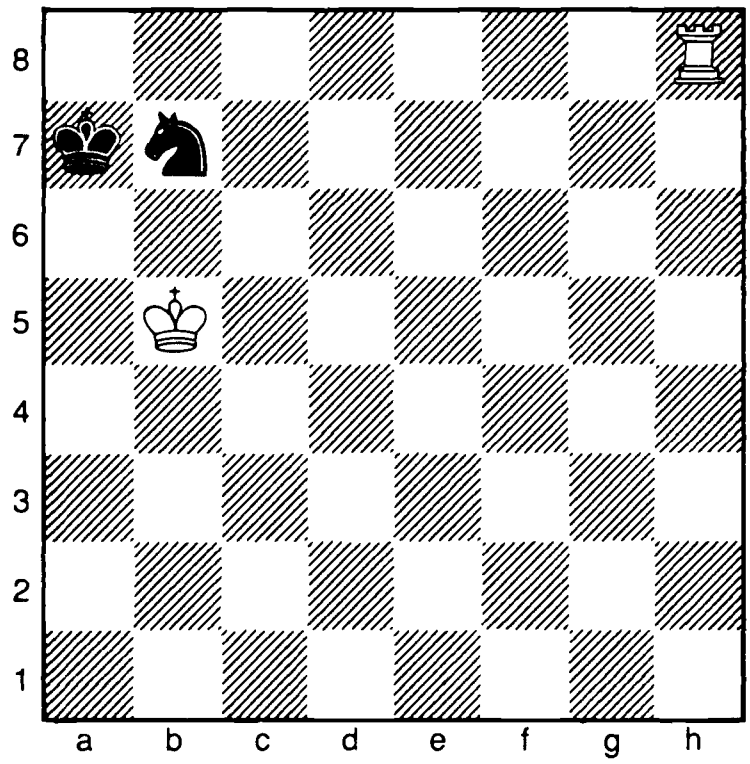
At any stage in this first part of the play Black will have a path back to the king for his knight. The trick is to ensure that this path leads to a tactical loss of the knight. The initial search is concerned with finding ways of forcing the knight away from the king and ensuring that return paths lead to a tactical loss.

There are several moves White can try. The simplest is a waiting move, which drives the knight to square d6. Rf8 is most efficient as it maximally constrains the knight after Nd6.

1. Rf8 Nd6

2. Kc6 ...

Figure 5.8



White to move

A variation of the al-Adli position
The minimax-optimal move is Rh5. The easiest move for a chessplayer to find seems to be Rf8 which is sub-optimal.

Attacking the knight and keeping the king trapped. Black's shortest path back to the king is Nb7 which loses to the pin Rf7, since White can force mate or material gain in a few moves. The next best path is Nc5, the only alternative being Ne4. We shall consider Ne4 later.

2. ... Nc4

Black now threatens Na5 and Ka6 to join the pieces. We are at the critical point in this variation. With Black to move we have 3. ... Na5, 4. Kb5 Nb7 and we are back in the al-Adli position with a move for the rook. The best place for the rook to be is d7, when the knight is paralysed. In fact most squares on the d-file are also good. To get the rook onto d7 we have to move Rf8 - f7 - d7 or Rf8 - d8 - d7. We also need to ensure that the knight moves to a5 rather than the king to a6. If the king moves to a6, rather than the knight to a5 and b7 a drawing pattern is set up. Thus the second path is best. Note that it is also possible to move the rook to the d-file with, Rf8 - f4 - d4. The square d4 is not as favorable as d7 however, as we shall see. We now have the variation,

3.	Rd8	Na5	
4.	Kb5	Nb7	back in the al-Adli setup
5.	Rd7	Kb8	Ka8 loses to mate threats
6.	Kb6	...	and White wins the knight

At this stage we have driven the knight from the king, although it is not completely separated in all variations.

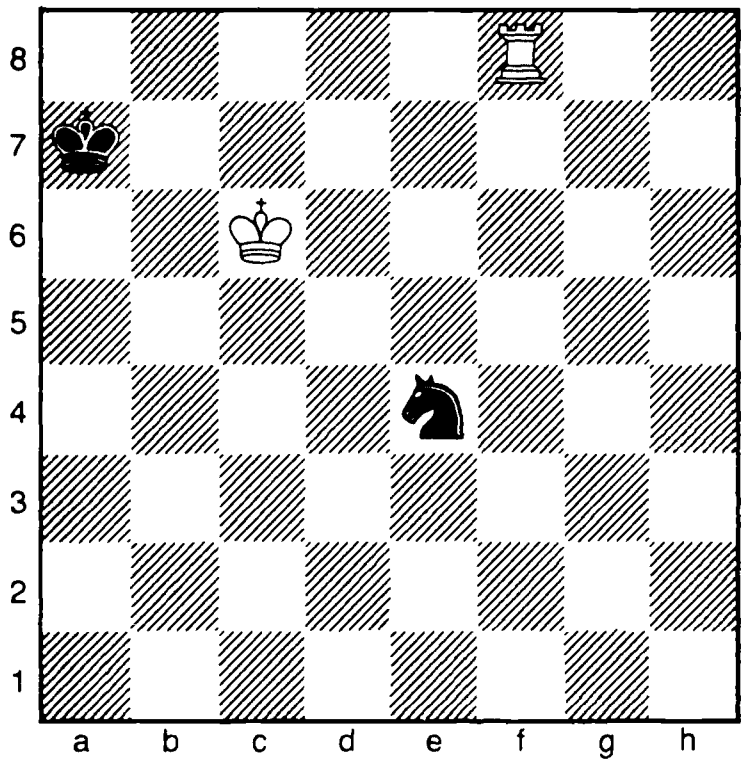
In each of the positions where Black can choose to separate the knight we have to ensure that this separation can be extended. Consider the position of Figure 5.9, which can occur in the main variation after 2. ... Ne4. The path distance of the knight from the king is 3. After 2. ... Ne4 the knight is quickly separated. One possible variation is,

3.	Rd8	Nc3	trying Na4 - b6
4.	Rd3	Nb1	
5.	Rc3	...	and the knight is trapped

Notice that if White had played 3. Rf4 earlier, instead of 3. Rd8 the trapping of the knight is much easier, while the refuting of Na5 is harder. The main line on Na5 is,

3.	(Rf4)	Na5	
4.	Kb5	Nb7	the al-Adli setup
5.	Rd4	Kb8	Ka8 loses quickly
6.	Ka6	Kc8	Nc5 loses to Kb6
7.	Rc4+	Kb8	
8.	Rb4	Ka8	now capturing the knight is stalemate
9.	Rh4	Nc5+	Kb8 loses to Rh8+
10.	Kb6	Nd7+	the only move

Figure 5.9



White to move

Black has played ...Ne4 and the knight
is now easily separated from the Black king.

11. Kc7 ... and the mate threat wins the knight

It is the difficulty of this line that explains why it is not considered by Berger or Fine in their analysis.

To summarise, an analysis of the al-Adli position shows that it can be solved by pursuing a series of subplans which lead to the capture of the knight. Each of these subplans is easy to execute. The simplest line found on this analysis is the one given in Fine's textbook, and turns out to be sub-optimal in the minimax sense.

5.7 Using Plans

I shall now consider in greater detail planning in positions where the the king and knight are separated. In particular positions where the knight is paralysed and cannot move without being captured. First let us outline the requirements for a system capable of formulating and reasoning about plans.

1. The system should be able to reason about a position in terms of plans and counterplans.
2. Plans should not be at the level of single moves, but should be at varying levels of generality.
3. The system should be capable of reasoning at the level of plans. It should be able in some circumstances to

refute a plan by, for example, demonstrating that its preconditions cannot be achieved rather than by search alone.

4. The system should be able to maintain the correctness of its plans. It should be able to prove that a plan succeeds.

5. Forcing trees have proved extremely useful, they should be retained in a form suitable for representing at least the lower level of plans.

One framework in which to phrase such a system is the generalised production system. The idea of a plan is intended to be a generalisation of the notion of better goal as used in the forcing trees we have been considering. A better goal specifies conditions on the board which need to be achieved, a plan specifies both the goal and the means for achieving the goal. One simple method of specifying the means to achieve a goal is to specify intermediate goals to be satisfied. The holding goals of a piece of advice satisfy this requirement in AL1 and Huberman's system but as a rather restricted limiting case. The notion of intermediate goals immediately suggests the hierarchical structuring of plans. Such a structure is well suited to proving properties of plans, very much in the way structured programs lend themselves to correctness proofs. The advantages of structuring robot plans in this way was first recognised by Sacerdoti (1977), although his NOAH system did not explicitly reason about plans, only about the order of actions within plans. To deal with plans and counterplans we need a broader definition of piece of advice. I give the definitions below, discuss them briefly and then

provide some illustrative examples in the KNKR domain, for positions with the Black knight paralysed. It should be noted that when we are dealing with plans and counterplans a proof that a particular plan succeeds is only conditional on the opponent using one of the counterplans considered.

Piece of Advice

A piece of advice for side X is a four-tuple (b_x, h_x, mc_x, mc_y) where b_x and h_x are predicates on positions and mc_x and mc_y are predicates on moves.
The quadruple (b_x, h_x, mc_x, mc_y) is written:

XBG: b_x
XHG: h_x
XMC: mc_x
YMC: mc_y

Note: the depth bound is subsumed within the move constraints.

Forcing tree

Given a position pos and a piece of advice $A = (b_x, h_x, mc_x, mc_y)$ a forcing tree for A in pos is a subtree of the game tree rooted in pos such that:

(1) for every node q in T, $h_x(q)$

(2) for every nonterminal node q in T, not $b_x(q)$

(3) for every terminal node q in T, either $b_x(q)$ or q is a Y-to-move position from which there is no legal move satisfying mc_y

(4) there is exactly one move in T from every X-to-move nonterminal node in T, this move

satisfies mc
x
(5) all legal moves from any nonterminal Y-to-move position in T satisfying mc are in T
y

Satisfiability

A piece of advice A is satisfiable in position pos if there exists a forcing tree for A in pos, written s(A,pos).
I use the convention that the argument referring to the current position in the tree is suppressed for goals occurring in a plan or piece of advice, e.g. the goal 'foo(q)' is written as 'foo'.

Plans

In Backus-Naur notation:
Plan ::= <b,h,m> | plan (; plan)* | <plan,h,b>

An X plan P = <b,h,m> succeeds for X in pos if s(A,pos) where A is:
XBG: b
XHG: h
XMC: m
YMC: nil

An X plan $P = \langle b_1, h_1, m_1 \rangle ; \langle b_2, h_2, m_2 \rangle$ with corresponding pieces of advice A_1 and A_2
succeeds iff s(A,pos) where A is:
XBG: $b_1 \wedge s(A_2)$
XHG: h_1
XMC: m_1
YMC: nil

An X plan P = <<b,h,m>,h',b'> is equivalent to a plan P' where
P' = <b/\b',h/\h',m>

Plans as defined above are an extension of the AL1 piece of

advice. The AL1 system provides only a fixed set of plans for an ending, and allows plans to be checked only by performing tree search. The definition of piece of advice and forcing tree given above is very similar to that used by the AL1 system, except that move constraints can be applied to either side.This is to allow consideration of counter-plans. Notice also that no restrictions are placed on the form of the various goals and move constraints. A better goal could for example be a recursive call to the same plan or piece of advice. An elementary algebra of plans is provided, which allows for the concatenation of plans (the ';' operator) and the restriction of plans via extra better and holding goals. Again we should note that the consideration of counterplans means that proof of the correctness of a plan is only conditional on the opponent executing some specified plan or plans.

A system to use this more general form of plan should contain a suitable control structure to generate and test plans and a knowledge base. This knowledge base will contain not only general information, but also specific facts about positions being studied. The following abilities can be provided.

1. Plans can be demonstrated satisfiable by direct tree search

2. Facts in the knowledge base can be used to provide a proof of the satisfiability or unsatisfiability of plans without search. Other strategies are also applicable, for example· if there are two possible subplans capable of achieving another plan, and both these plans share the same initial move(s) this move

can be made without further search.

3. Knowledge in the knowledge base can be used to refine plans, reducing search.

4. The knowledge base can be used bottom up to generate plausible plans from high level descriptions of positions.

5. Information gained from the failure of plans can be passed, via the knowledge base to permit refinement of plans.

6. Instead of performing search directly plans can be passed across moves to form an and-or tree of plans. For example if $P_1, \ldots, P_n$ are successors of position P then plan will succeed in P iff Plan$_1$ \/ ... \/ Plan$_n$ succeeds in $P_1, \ldots, P_n$. In this way the knowledge base can be used at each position during forcing tree generation. This also permits strategies other than depth first for searching.

The following definitions give a formal specification of the interactions between plans, in particular what it means for one plan to refute another. In addition we define the concept of knight paralysis and macromove more formally. Explanations are given for some of the definitions.

Def Plan pseudosucceeds

X plan P = <b, h, m> pseudosucceeds in pos iff
s(A,pos) where A is:

XBG: b
XHG: h
XMC: m
YMC: dummymove (no move)
Explanation A plan P pseudosucceeds if the plan can be executed by X with Y making no move.

Def P_y refutes P_x

Let X plan be $P_x = \langle b_x, h_x, mc_x \rangle$ and Y plan be $P_y = \langle b_y, h_y, mc_y \rangle$, then plan P_y refutes P_x in pos iff not s(A,pos) where A is:

XBG: b_x or not h_y
XHG: h_x
XMC: mc_x
YMC: mc_y

Explanation P_y refutes P_x if P_x cannot succeed while Y is executing P_y.

Def Pseudolegal

For an X macromove M and a position pos
pseudolegal(M,pos)
iff s(A,pos) where A is:

XBG: final state of M
XHG: true
XMC: M
YMC: dummymove

We also need some definitions of terms,

Def Path

(Piece)path(p) <=> $p = \langle s_1, \ldots, s_n \rangle$ and
(piece)dist(s_i ,s_{i+1}) = 1 for i = 1, ..., n-1

Def Length

length(p) = n <=> $p = \langle s_0, \ldots, s_n \rangle$ for some $s_0, \ldots, s_n$

Def King macromove

K S1 S2 (d) = {p: kingpath(p) and length(p) = d and
p = <S1 ,... ,S2> }

Def $S_N(P)$, $S_W(P)$, $S_R(P)$, $S_B(P)$ are sets of
squares attacked by the BN, WK, WR and BK respectively.
The piece does not necessarily have a legal move
to any of these squares.

Def Rsafe(P) <=> s(A,P) where A is:

WHG: not(rook captured)
BMC: depth less than or equal to 2 ply

Def Nsafe(P) <=> s(A,P) where A is:

BHG: not(knight captured or Black mated)
WMC: depth less than or equal to 2 ply

Def Nparalysis

Nparalysis(P) <=> $S_N \subseteq S_W \cup S_R$ and $S_N \cap S_B \subseteq S_R \cap S_W$
and rsafe(P)

The following are some of the productions in the knowledge base. Productions are divided into two classes, Lemmas which are used as facts and Methods which are used to control reasoning. A method consists of a left hand side (LHS), a right hand side (RHS) and a (possibly empty) set of logical relations. These logical relations assert facts about hypotheses which are to be investigated. A method can fire when the predicates on its LHS are true and when the hypotheses on its LHS are current hypotheses. When a method fires the hypotheses on its RHS are added to the current set of hypotheses. Predicates are distinguished from hypotheses by being underlined. Hypotheses are tagged with identifiers of the form

'Hi' to aid in identification, these tags are not part of the hypotheses.

method NP (Nparalysis)

Nparalysis --> H: <Nwon, nil, nil> succeeds

method WINN1 (Win N by K macromove)

Nparalysis and H: <Nwon, nil, nil> succeeds -->

H1: <Nwon, Nparalysis, KxN> succeeds
(H1 => H)

method SPLIT (Split plan and refute its parts)

H: <b, h, M> succeeds -->

H1: <b, h, M> refutable by RM

H2: <b ,h, M> refutable by RH
(H1 or H2 => not H)

method RM (Refute WK macromove)

H: <W S1 S2 (d)> refutable by RM -->
(Let P be <W S1 S2 (d)>)

H1: P refutable by RM1

H2: P refutable by RM2

H3: P refutable by RM3
(H1 or H2 or H3 <=> H)

method RM1 (Refute K macromove by K macromove)

H: <b, h, XK S1 S2 (d)> refutable by RM1 -->

H1: <YK b S (d)> refutes <XK S1 S2 (d)>

H2: pseudosucceeds(<XK S1 S2 (d)>)
(H1 => H)
(not H1 and H2 => not H)

lemma RM1B (Refutability of WK macromove)

H: <B b S> refutes <W a z (d)> -->

H1: pseudolegal(W a z (d))

H2: d(z,b) < d or BTM and d(z,b) < d+1
(H and H1 => H2)

lemma LPW ('Piece win-able')
H: <M, Piece-paralysis, Piece-won> succeeds -->

H1: <M, Piece-paralysis> refutable by RM

H2: <M, Piece-paralysis> refutable by RH
(not(H1) and not(H2) and (Piece-square is target of M) => H)

method PLANPS (Plan pseudosucceeds)
H: pseudosucceeds(<b, h, M>) -->

H1: s(A,pos) where pos is the initial position of M, X is the side that executes M and A is:

XBG: b
XHG: h
XMC: M
YMC: dummymove
(H1 <=> H)

method SEARCH

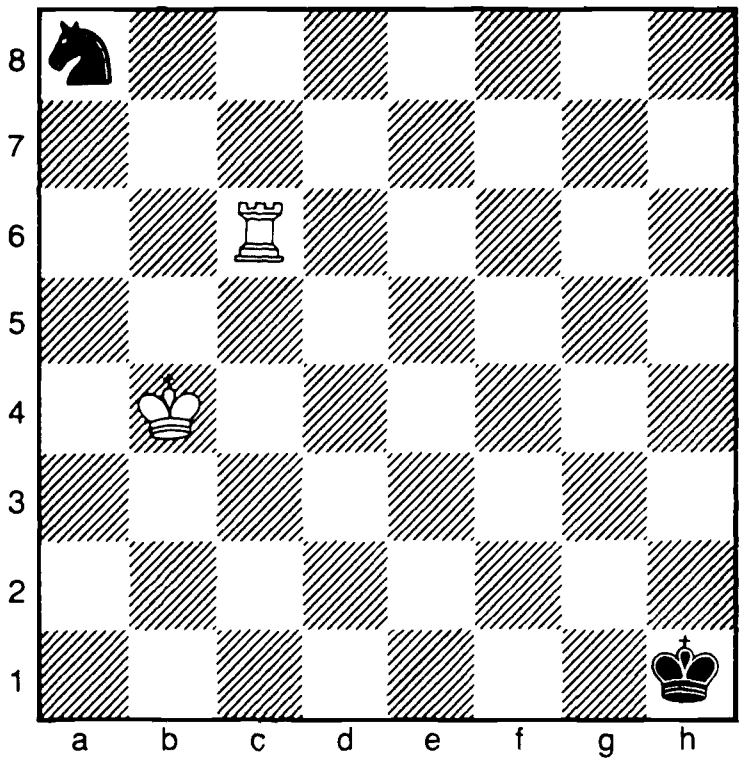
H: s(A,pos)

Do the game tree search to see whether A is satisfiable in pos. If yes, then return the forcing tree. If no then return the refutation tree.

Consider the position of Figure 5.10. The process of plan generation and confirmation is outlined below.

method NP Nparalysis -->

Figure 5.10



White to move

The Black knight is paralysed and can be captured by the White king.

H1: <Nwon, nil, nil> succeeds

method WINN1 Nparalysis and H1 -->

H2: <Nwon, Nparalysis, KxN> succeeds
(H2 => H1)

method INST1 H2 -->

H3: <Nwon, Nparalysis, W b4 a8 (4)> succeeds
(H3 => H2)

Let M be W b4 a8 (4) and P be <Nwon, Nparalysis, M>

method SPLIT H3 -->

H4: P refutable by RM

H5: P refutable by RH
(H4 or H5 => not H3)

method RH H5 -->

H6: P refutable by RH1

H7: P refutable by RH2

H8: P refutable by RH3
(H6 or H7 or H8 <=> H5)

lemma RH1b H6 -->

H9: d(h1,c6) > 4 and d(h1,{c7,b6}) > 3

H10: pseudosucceeds(M)
(H9 and H10 => not H6)

method METRICS H9 is true

method PS H10 -->

H11: s(A,pos) where A is:

WBG: W at a8
WHG: Nparalysis
WMC: W b4 a8 (4)
BMC: dummymove
(H11 <=> H10)

method SEARCH H11 is true, therefore H6 is false.
Similarly H7 and H8 are false.

Following implications, H5 is false

method RM H4 -->

H12: P refutable by RM1

H13: P refutable by RM2

H14: P refutable by RM3
(H12 or H13 or H14 <=> H4)

method RM1 H12 -->

H15: <B h1 S (< 4)> refutes <M>
(H15 => H12)
(not H15 and pseudolegal(M) => not H12)

lemma RM1B H15 -->

H16: pseudolegal(M)

H17: d(a8,h1) < 3 and WTM and d(a8,h1) < 4
(H15 and H16 => H17)

method MACROPL H16 -->
H18: s(A,pos) where A is:

WBG: W at a8
WMC: W b4 a8 (4)
BMC: dummymove
(H16 <=> H18)

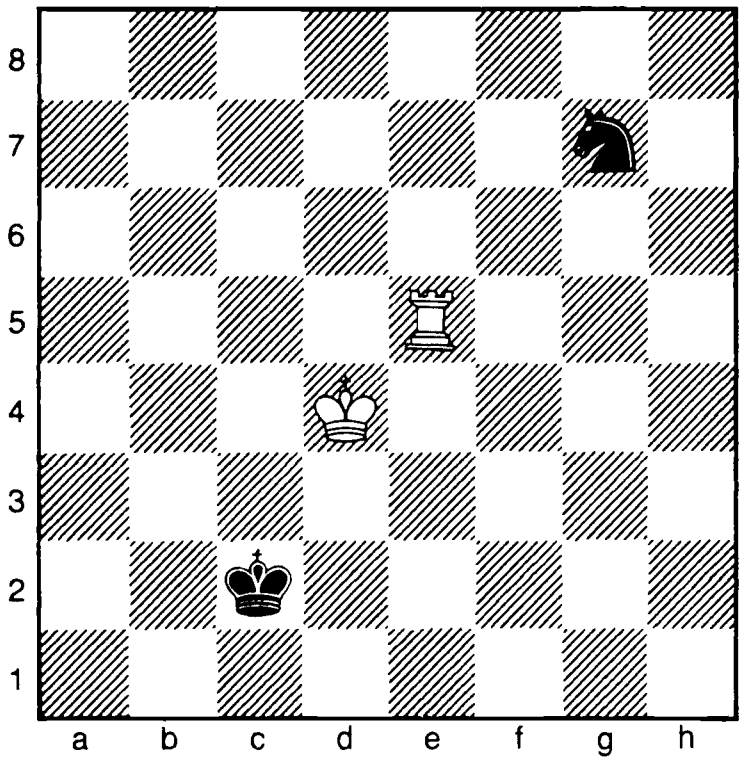
method SEARCH H18 is true, therefore H12 is false.
Similarly H13 and H14 are false.
Following the implications, H4 is false.

lemma LPW (a8 is target of M) and H4 is false and

H5 is false => H3 is true => H1 is true.

For our second example consider the position of Figure 5.11. This example is more complex than the previous one, and details are omitted. Simple though the position is the minimum depth of

Figure 5.11



White to move

The Black knight is paralysed and can be captured by the White king. The main variation is:
1. Ke4 Kc3 2. Kf3! Kd4 3. Kf4 Kd3 4. Kg5 Kd4 5. Ke7 Kd5 6. Rxg7

capture of the knight is 11-ply. The main variation goes,

1.	Ke4	Kc3
2.	Kf3!	Kd4
3.	Kf4	Kd3
4.	Kg5	Kd4
5.	Re7	Kd5
6.	Rxg7	

Notice that a 4 ply search would not discover the zugzwang position with the White king on f4, the Black king on d4. The problem we face here is of finding the appropriate plans for White and Black. The initial plan we produce for White is P = <KxN, nparalysis, W d4 g7 (5) >. It is necessary to reduce the amount of search done considering moves for Black. The simplest solution is to consider only sets of squares on which the Black king can be. In this manner there is no need to constrain the king moves since only squares on which there is a direct interaction between the pieces will be considered. Similarly for the White king we need consider only the square sets of the White king macromove. We define for M = <K S1 S2 (N)> the sets M(i) i=0,...,N to be the set { s: s is on the ith move of some path in M}.

With this definition the search for the initial plan shows that the plan fails since White cannot move from M(2) to M(3) without violating the holding goal of nparalysis. The structure of

the refutation tree allows us to use a causal method to produce a new plan from the ruins of the old. This method notices that at the refutation node if White makes a dummy move, after Black's move White can proceed with his plan. The causal analysis then suggests a new plan for White consisting of the macromove K d4 g7 (5). This process is repeated several times, until we reach a main variation,

1.	Ke4	Kc3
2.	Kf3	Kc4
3.	Kg4	Kd4
4.	Kf4	Kd3
5.	Kg5	Kc4
6.	Kg6	Kd4
7.	Kf6	Kd3
8.	KxN	

Thus a very deep solution is found without excessive search. One form of a search algorithm using move sets is discussed in the next chapter.

The greatest difference between the search behavior of AL1 and this suggested system is the dynamic creation of plans, which allows a very flexible approach to the problems of positions. Other differences spring from this. No attempt is made to prove the global correctness of the system for the ending. It is

certainly possible to specify a few general plans at the highest level which exhaust the space of plans overall. The system is left to fill in the details, and to ensure that the plans found are correct. We should note that the type of search necessary in positions such as that in Figure 5.8 requires a great deal of knowledge about the value of individual positions and classes of position, and about the appropriate plans for each side to follow in these positions. An inductive facility is invaluable here to enable the expert user to train the system with examples of won and lost positions, and for the system to be able to classify such positions.

There are two major problems to be faced when we use plans in the manner indicated above. The first is that of completeness. The use of a plan for one side is essentially the use of meta-knowledge about the object level tree search, stating that in certain conditions a restricted search will produce the same results as an unrestricted one. An AL1 forcing tree can be considered as a plan for only one side. If a piece of advice is satisfiable we are guaranteed that the unrestricted search would also succeed. On the other hand if a piece of advice is unsatisfiable we can in general say nothing about the success of the unrestricted search. This is no problem because we can try different plans, the whole set of which will be complete (completeness to be proven by the expert providing the plans). If any plan succeeds, the unrestricted search would succeed. If no

plan succeeds the unrestricted search would fail. Things are not so simple if plans are used for both sides. It is possible for one plan to succeed against another even though the unrestricted search would fail. In these circumstances it is necessary for the individual plans to be complete. This is hard to achieve.

To sum this up; if we to do very deep searches, which seem necessary for complex endgames such as KRKN, we cannot hope to consider every possible sequence of moves for the opponent. What we can hope to achieve is consideration of a sufficiently rich set of plans for the opponent that will encompass all of his possibilities for refuting our plan. We must note however that unless we can prove that the counter-plans considered are complete in this sense, we will not be proving that a plan succeeds but rather that it succeeds subject to the opponent following certain plans.

The second problem arises from the relative intractability of plans to decomposition. For example if goal A is unachievable and goal B is unachievable, it is possible that the disjunction of the two goals is achievable. This particular problem is met in the next chapter.

In the next chapter we consider the endgame of King and Pawn vs. King with a view to investigating methods of doing deep searches within relatively unconstrained plans, and to providing a

hierarchical structure of plan synthesis to allow the incorporation of heuristics to control search, without affecting the validity of strategies.

6. An advice strategy for KPK

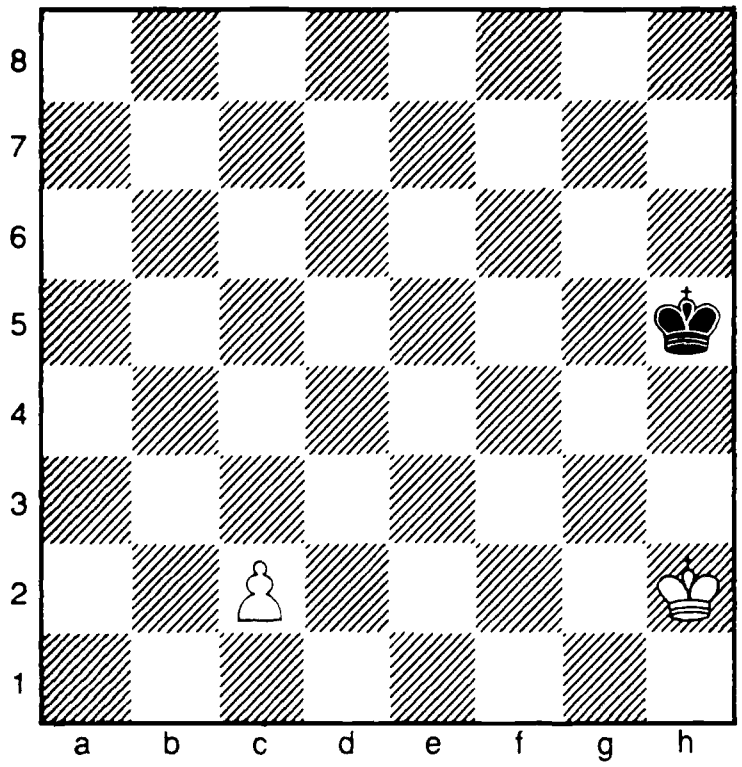
6.1 Introduction

This chapter describes a strategy for the endgame King and Pawn vs. King (KPK), and its implementation. Consideration is given to the knowledge representation issues involved, and to methods by which the strategy can be demonstrated correct.

Although one of the most elementary chess endgames, KPK is not trivial for the inexperienced player. There are many positions where the correct move can only be found by subtle reasoning and considerable use of domain-specific knowledge. Two examples are given in Figure 6.1 for the reader to consider before a more detailed discussion is given.

The KPK endgame is highly resistant to conventional programming techniques. It can take months of programmer effort to implement and then it is extremely hard to produce a correct program. Kopec (1979) details some of the problems. The endgame has been studied extensively. Decision rules for classifying positions have been produced by Beal (1977) and Niblett and Shapiro (1980). Knowledge-based playing programs have been written by Tan (1972), Harris (Kopec 1979) and Bramer (1980b). Bramer has produced both a minimax-optimal program validated by exhaustive enumeration of the move played from each position, and

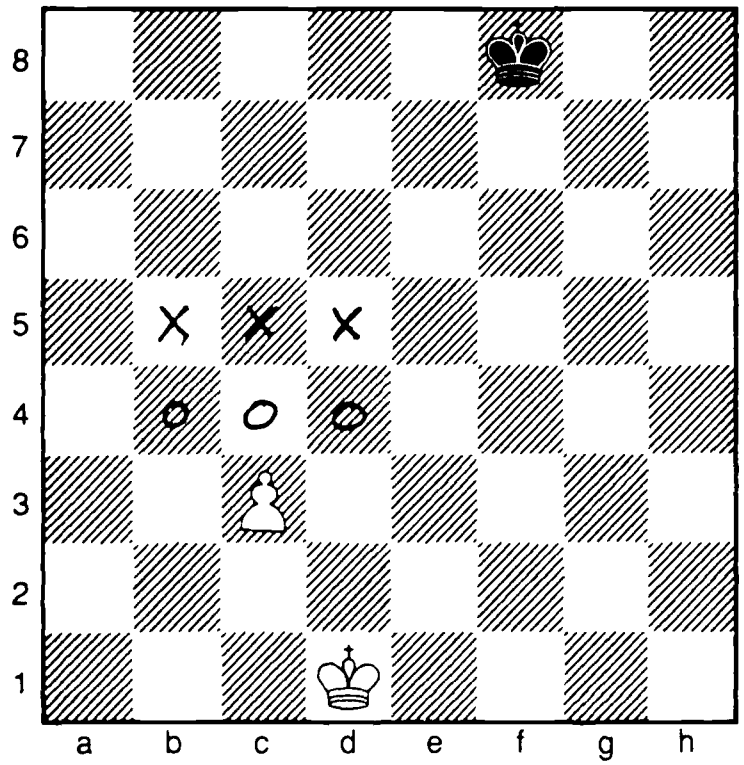
Figure 6.1a



White to move

White's only winning move is Kh3. This position is one of the deepest (38 ply) wins in the KPK space.

Figure 6.1b



White to move

White's only winning move is Kc2.

a correct program validated by a backing-up algorithm (Bramer 1980a, Bramer 1980b).

In this chapter the principles of the endgame as given in chess textbooks are presented. The playing strategy produced is described and evaluated. A method of implementation, designed to reduce the burden of performing deep searches is described.

6.2 The play of King and Pawn vs. King

Our description of KPK is based on the lucid exposition in 'Pawn Endings' (Averbakh & Maizelis 1974), the standard textbook for the ending.

The knowledge embodied in the text can conveniently be divided into two categories:

1. General strategic principles describing the structure of the ending and the goals for the side with the pawn (here taken to be White).
2. Specific tactical examples to illustrate the application of these principles and to note any exceptions which may occur.

My aim in the development of a playing strategy was to reflect this separation of knowledge into two categories and to investigate its transcription into knowledge structures in a

program. At the same time attention has been paid to the nature and extent of additional knowledge that must be added due to insufficient specification in the text.

Below is a brief overview of Averbakh's description of the ending, including both the strategic and tactical components. We describe ways in which his description is inadequate for a playing program and present the advice language strategy.

The play of KPK is determined by two considerations:

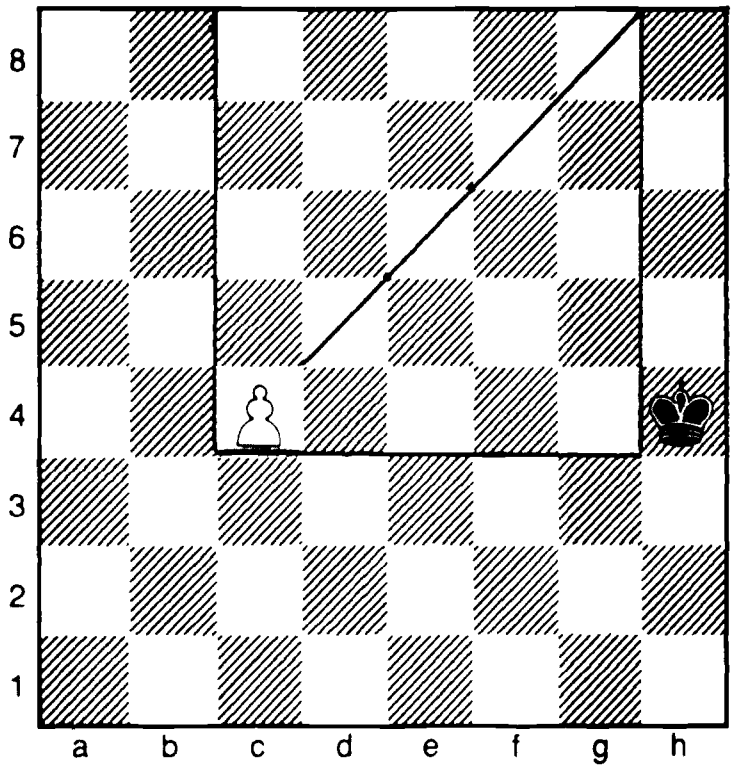
- The square of the pawn, and
- The pawn's critical squares

Figure 6.2 illustrates the square of the pawn.

With White to move (WTM) the pawn advances, the square contracts and the Black King cannot move into the new square, allowing the White pawn to safely queen. With Black to move (BTM) the Black king can enter the square and so prevent the pawn queening.

The critical squares of the pawn are shown in Figure 6.3. When the pawn's rank is less than 5 the critical squares are shown in Figure 6.3a, otherwise as in Figure 6.3b. If the White king can occupy one of the critical squares White wins whoever is to move.

Figure 6.2



White to move

The 'square' of the pawn.

Figure 6.3

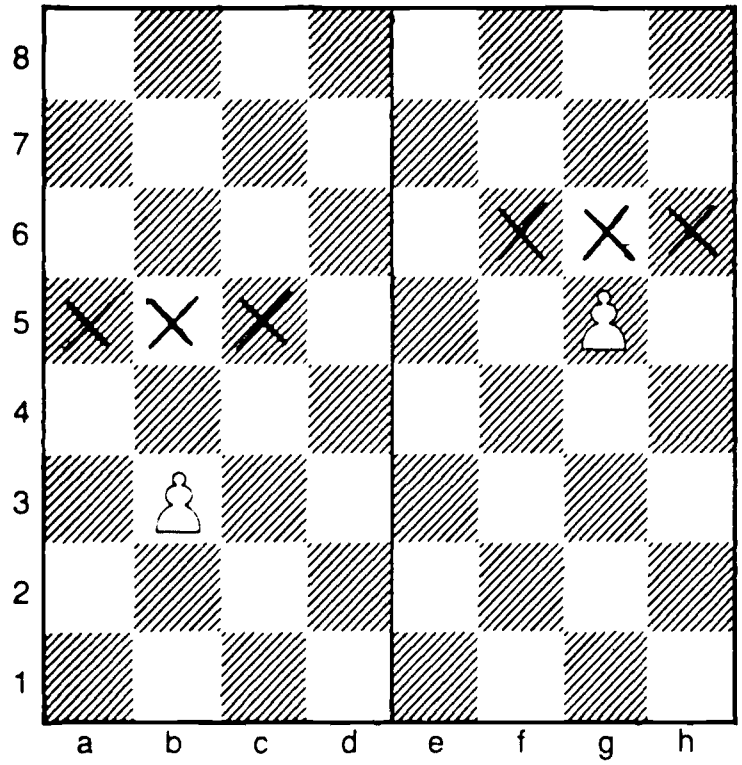


Figure 6.3a

The critical squares are 2 ranks ahead of the pawn

Figure 6.3b

The critical squares are 1 rank ahead of the pawn

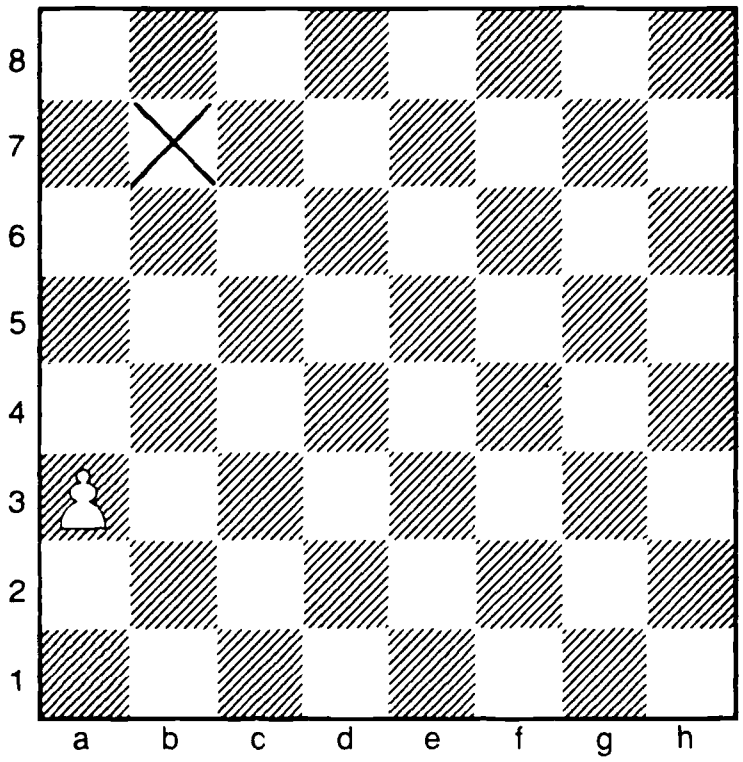
Averbakh treats the rookpawn as a special case, with only one critical square, the location of which is independent of the pawn, shown in Figure 6.4.

These two principles, of the pawn's square and occupation of the critical squares provide a broad orientation, or strategy, for the player with the pawn. By its nature the tactical component is impossible to summarise concisely. Averbakh cites about a dozen examples of play occupying several pages of text. Figure 6.1b is an often cited example.

In this position White can force occupation of one of the pawn's critical squares (marked with an x), by the move sequence Kc2 - b3 - b4 and thence either gain the opposition on c4 or move to b5. Any other path for the White king fails. This position is intended to illustrate the heuristic of moving the White king to the opposite side of the pawn from the Black king if possible.

It is worth discussing the example at slightly greater length, because it demonstrates the difficulty of representing this type of knowledge in a program. A direct translation of the heuristic is inappropriate because it lacks generality. The solution to the problem of White's forcing his king onto one of the critical squares is essentially geometric. From the 'theory' of the opposition we know that if the White king can reach one of the squares marked 'o' without Black achieving the opposition he

Figure 6.4



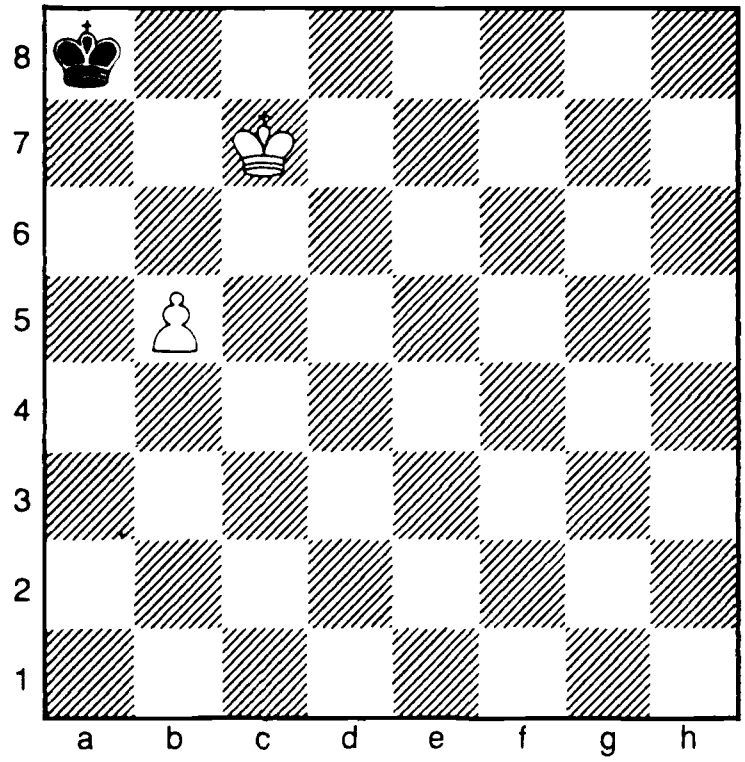
The critical square for the rook's pawn

can reach one of the critical squares. By geometric considerations we see that following the path d1 - c2 - b3 - b4 White can reach a square marked 'o' without Black being able to force the opposition since in the starting position he is already 2 files to the right of the White king. The problem with the original heuristic is that it is applicable in only a limited class of positions, and it is hard to produce a description of exactly which positions these are.

In a later section dealing with implementation of the advice strategy we return to this position and show that this type of elementary geometric consideration can be implemented by choosing a suitable representation for the search strategy.

There is another major distinction that can be drawn within this tactical knowledge about the domain. This is knowledge about exceptional positions· where the normal search strategy is inadequate. An example is shown in Figure 6.5. The standard procedure for White in positions of this type is to advance the pawn, thereby assuring that he will be able to occupy one of the pawn's critical squares with the pawn further advanced. In this case however advancing the pawn leads to stalemate and White has to pursue an alternative plan viz. Kc7 - b6 - a6, followed if possible by Ka7 else by Pb6. The important point here is that we must realise that the tactical plan to achieve our strategic goal (of advancing the pawn) fails, and that another tactical plan for

Figure 6.5



White to move

Advancing the pawn produces stalemate with this configuration of the pieces.

the same strategic goal should be applied. As we shall see this dual level of structure enormously simplifies the programming of the ending. This example shows a strictly pattern-based approach to be more unwieldy as the number of patterns in the knowledge base can grow large. An element of search seems convenient for a compact representation of the ending.

The textbook representation of the problem domain is lucid to the human player. For a playing program it is deficient in several respects.

Necessary background information is omitted, usually because it is too obvious. For example Averbakh states that positions of the type shown in Figure 6.3 are won for White with either side to move. He does not explicitly allow for the possibility that the Black king on move may be able to capture the pawn, precisely because it 'goes without saying'. When writing a playing program, or trying rigorously to prove the correctness of a playing strategy these considerations have to be taken into account!

The tactical information supplied is not precise enough for a formal description of the search strategy to be used in achieving the strategic goals. Again a human player is expected to use his more general chess knowledge to fill in these gaps.

From these considerations we can conclude that a major task

of the advice strategy designer is to use the knowledge representation tools at his disposal in producing a formally defined strategy which is correct. Indeed it is of considerable interest to see exactly how much knowledge is needed for a complete specification of play in the ending.

6.3 The Advice Strategy

The 'advice text' given below is a (rough) English translation of the advice strategy. The strategy itself is written as a set of logical rules. The notion of controlling the critical squares of the pawn is captured by the pattern mainpatt, which is a generalisation of the pattern of Figure 3. The notion of the 'pawn's square' is captured by the predicate canrun which holds whenever the pawn can promote without the White king having to move. A special pattern rookpatt is used in the rookpawn case.

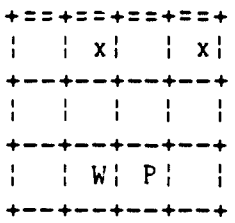
If the strategy is applied to a won position a plan is returned that wins for White. If the strategy is applied to a drawn position then no plan will be returned. Any move will preserve the game-theoretic value in this case. Note that where a compound plan of the form (Plan1 ; Plan2) is returned the second constituent of the plan is a function (apply-rule) which returns a possibly underdetermined plan. The exact plan to be followed depends on Black's moves. What we have is a recursive call to the advice strategy. Note also that in rule 3.2 the double implication arrow means that if this rule fails for a rook pawn then there is no winning plan for White, and the other rules

should not be attempted. We again use the convention that the argument for the current position is suppressed inside plans. It would perhaps have been closer to accepted practice to use a 'dot' convention for the missing argument, e.g. rank(p,.), but it is hoped that what we have done will be clear enough.

AN ADVICE TEXT FOR KPK

To determine whether White wins from any position, consider the following rules in order:

- If the pawn can run White wins
- If the pawn is a rookpawn White wins iff a position is achievable where the pawn can run or where rookpatt holds
- If the pawn's rank is 7 White wins if the White king can safely move next to the queening square
- If the pawn's rank is 6 and the pattern shown below holds White wins



Black king on any but X

- If the pawn's rank is greater than or equal to 5 then White wins if he can achieve mainpatt or the pattern of the previous rule.
- If mainpatt holds then White wins
- White wins iff mainpatt or canrun is achievable

AN ADVICE LANGUAGE STRATEGY FOR KPK

Rules

```
1)  wins(white,P) <--  wins_with(white,P,S).

2) wins_with(white,P,<kdist(p,qsq(p))=0,pawnsafe,if stalepatt then
                                                    kingmove
                                                 else
                                                    pawnmove>)
                                    <--  canrun(P).

3.1) wins_with(white,P,rookplan) <-- rookpatt(P).

3.2)  (wins_with(white,P,
          <<canrun \/ rookpatt,pawnsafe,nil>;apply-rule>) <-->
          s((canrun\/rookpatt:pawnsafe:nil:nil),P))
                  <-- rookfile(p,P).

4)  wins_with(white,P,
           <<kdist(w,qsq(p))=1,pawnsafe,kingmove>: apply-rule>)

  <-- rank(p,P)=7/\s((kdist(w,qsq(p))=1:pawnsafe:kingmove:nil),P).

5)  wins_with(white,P,<<nil,nil,push(p)>;apply-rule>)

                              <--  pattern_rank6(P).

6)  wins_with(white,P,
          <<pattern_rank6\/mainpatt,nil,nil>;apply-rule>)

     <--  rank(p,P) >= 5 /\ not(mainpatt(P)) /\
         s((pattern_rank6\/mainpatt:nil:nil:nil),P).

7)  wins_with(white,P,<mainplan;apply-rule>)
                    <--  mainpatt(P).

8)  wins_with(white,P,<<mainpatt\/canrun,pawnsafe,nil>
                                                    ;apply-rule>)

     <--  s((mainpatt\/canrun:pawnsafe:nil:nil),P).
```

NOTATION AND DEFINITIONS

Some of these definitions were given in more detail in section 5.7.

w,b,p — denote the White king, Black king and White pawn respectively.

P — The current (White-to-move) position

Plans

A plan is a specification of a strategy for one side. The Backus-Naur formulation is:

Plan ::= <Better-goal,Holding-goal,Move-constraints> |
<Plan;Plan>.

(Note: the fuller definition of plans given earlier in chapter 5 is not needed here)

Advice

A piece of advice is a quadruple:

(better goal:holding goal:W move constraint:B move constraint)

Satisfiability

A piece of advice A is satisfiable in position P (s(A,P)) iff there exists a non-empty forcing tree satisfying A rooted in P.

wins(Side,Position) iff side Side can force a win from Position.

wins-with(Side,Position,Plan) iff Side wins from Position
with Plan.

apply-rule(P) is a function which returns a (possibly
underdetermined) plan found by entering
the advice strategy with position P.

wtm(P) iff White is to move in P.

btm(P) iff Black is to move in P.

qsq(p,P) The queening square of the pawn in P.

stalepatt(P) iff promoting the pawn to a rook or queen
leads to stalemate.

canrun(P) iff s((kdist(p,qsq(p))=0:pawnsafe:pawnmove:nil),P).

pawnsafe(P) iff wtm(P) or Black cannot legally capture the pawn.

rank(Piece,Position) is the rank of Piece in Position.

kdist(Piece,Square,Position) is the number of kingmoves (not
necessarily legal) from Piece to Square in Position.

push(Piece,Pos) is the move of Piece 1 rank up the board in Pos.

pattern-rank6(P) iff the White pawn is on the 6th rank with the White King next to it on the same rank, and the Black king is not on the 8th rank 1 file distant from the pawn.

rookpatt(P) iff the White pawn is on the rook's file and the White king is on some other file, on the same rank as the Black king, and nearer the pawn (filewise) with Black-to-move.

rookfile(p,P) iff the pawn is on the rook's file in P.

mainpatt(P) iff wtm(P), the pawn is not on the rook's file, the White King coordinates relative to the pawn are (-1,2),(0,2),(1,2) or (-1,1),(0,1), (1,1) and the two kings are not in direct opposition. Also Black must not be in stalemate in P.

mainplan is the plan of achieving mainpatt with the pawn further advanced. More formally it is given as:

<mainpatt, pawnsafe, rank(p) > rank(p,P)>

rookplan is the plan:

```
<kdist(p,qsq(p))=0, pawnsafe, if  rank(w) < 7 and
                                  rank(w) < rank(b)

                              then
                                      push(w)
                              else
                                      push(p)>
```

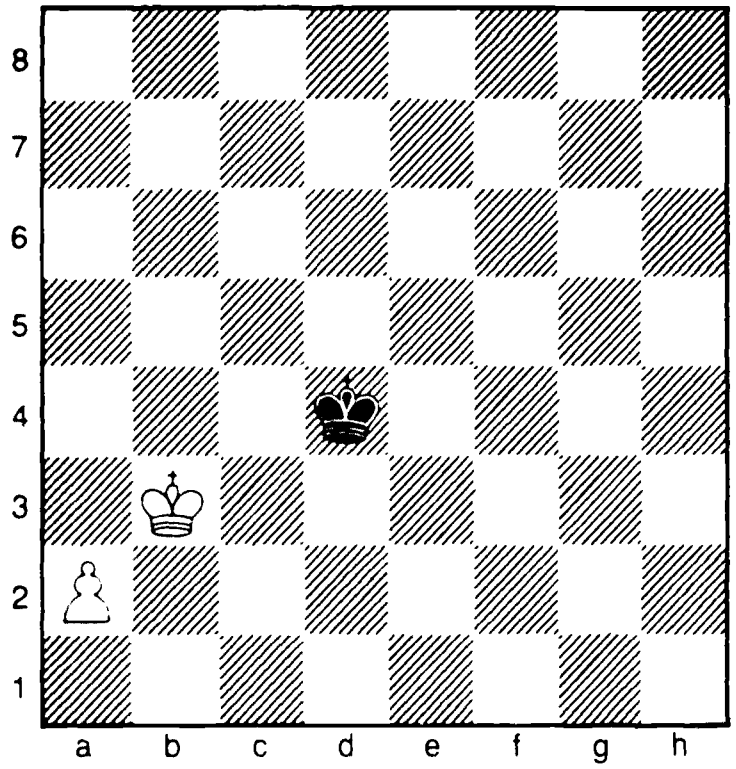
The proof of correctness is given in Appendix 3. I shall make a couple of points about it. It is necessary to show that

1. White wins from any of the patterns on the right hand side of a rule. The main burden of this half of the proof is to show that from any position satisfying mainpatt White can force mainpatt with the pawn further advanced, or promote the pawn
2. If White can win, one of the conditions on the right hand side of a rule is true.

The proof is straightforward though long . Care must be taken in dealing with exceptional positions. For example it is true that if the pawn can run White wins, but not true that White can always win by promoting the pawn. There is one exceptional position where any promotion that could win causes stalemate!

The rule which caused most trouble to produce was for the rookpawn. The textbook contains very few examples of play and the strategic rule is vague. A detailed analysis of critical positions had to be made before a rule which produced acceptable search characteristics was found. Figure 6.6 shows examples of difficult

Figure 6.6a



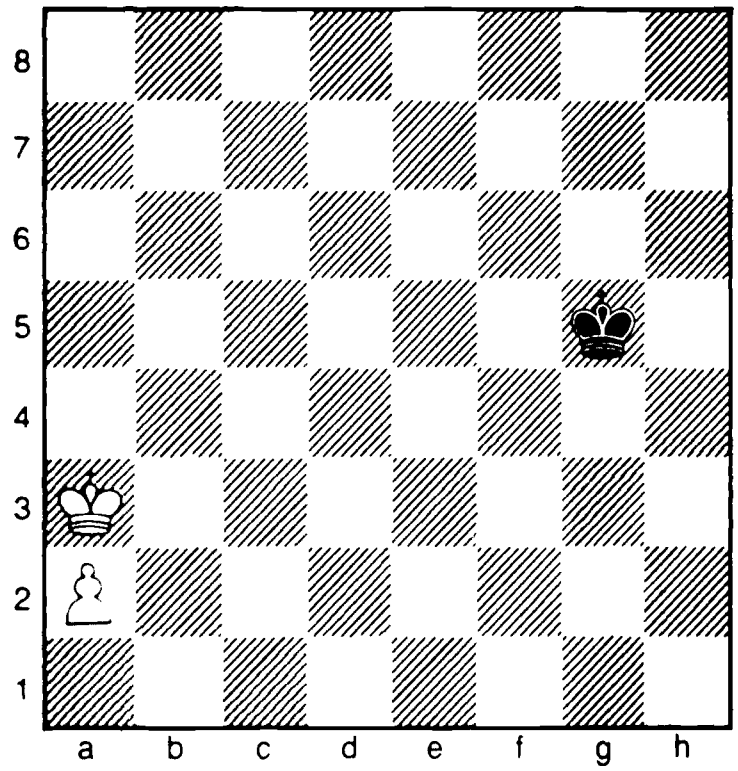
White to move

The main variation ensuring that the pawn can run is,

1. Kb4 Kd5 2. Kb5 Kd4 3. a4 Kd5

4. a5 Kd6 5. Kb6

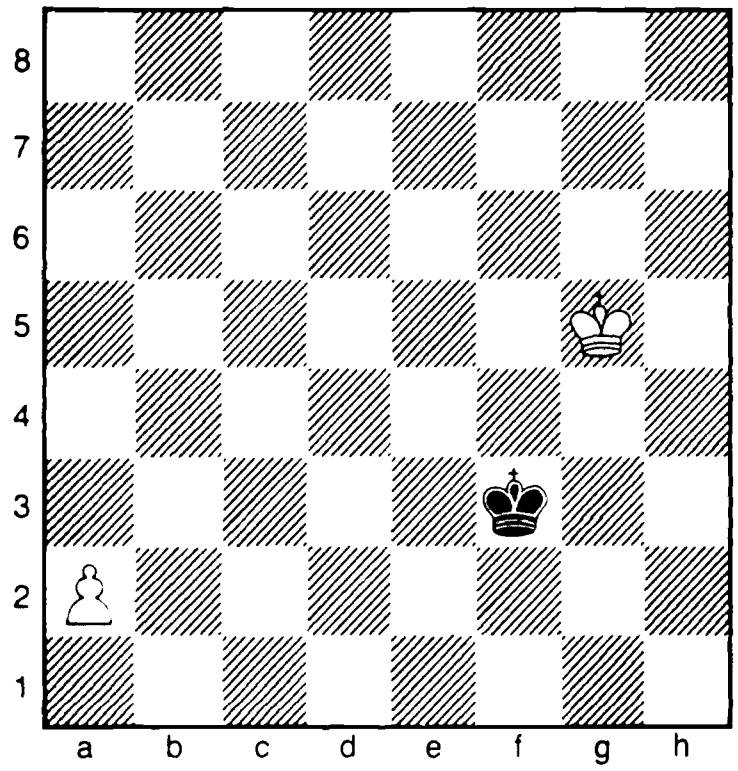
Figure 6.6b



White to move

Kb4 wins for White

Figure 6.6c



White to move

Kf5 wins for White

positions with the pawn on the rook's file.

The rule settled on is that White should manoeuver for a position where the pawn can run. There are two problems with this rule. Firstly, the search is too deep, and secondly it is not trivial to recognise CANRUN positions statically. Figure 6.6a demonstrates the force of the first objection. A 9 ply search is necessary before a position is reached where the pawn can run. It is easy to extend the rule to make such positions terminal. This reduces the maximum depth of search to 7 ply. There are fewer than twenty positions in the position space that require such a deep search. These fall into 2 classes, illustrated in Figure 6.6b and 6.6c. The first class is of positions where the two kings race for the square b7, the second where the White king must maintain opposition above the Black king until the pawn can run. This search is tractable when using the square sets algorithm. The second problem, the static description of the CANRUN predicate, is also tractable when the search uses square sets.

6.4 Search

The advice strategy and its proven correctness formalises the first part of Averbakh's description of KPK. The strategic foundations have been laid. The strategy has now to be fleshed out with sufficient extra knowledge to evaluate each method

efficiently. Before this is done we need to know how much search the various rules require. Search is necessary to match the LHS of rules when the satisfiability of pieces of advice is involved. I will address three questions:

1. How much search does each rule require?
2. What extra knowledge must be added to make the necessary search small enough?
3. Which search space representation is most appropriate?

These three questions are related and logically 1) and 2) are dependent on 3). Before fixing 3) however some rough measurements can be made of 1).

A broadly effective measure of the search required by a rule is the maximum search depth required by that method. For a wide variety of search algorithms the amount of search increases exponentially with depth. There are two ways of determining the maximum search depth for the various methods in the advice text:

1. Refinement of the correctness proof to provide an upper bound on search depth.
2. The use of a database method.

The second option was chosen here. Databases were constructed for each piece of advice. The algorithm used is described in Appendix 2. The maximum search depth is shown in the table of Figure 6.7. From this table we can see that rule 7 (for example) with a maximum search depth of 9 ply will need extra search information.

Figure 6.7

Rule Number	Maximum Search Depth
2	9
3	7
4	3
5	0
6	5
7	9

Maximum depth of search for every

rule in the advice strategy

Note that the maximum depth is always needed to establish unsatisfiability, but less will be needed to establish satisfiability in general. Extra knowledge can be added to reduce search, but this may affect correctness.

What is the nature of this information? Several points should be made:

1. It is subordinate to the main strategic rules, giving information as to how these goals are to be achieved.

2. It is not necessarily fixed and immutable within the system, and can be extended if it is thought desirable.

3. It can be split into separate components. Firstly knowledge to order the consideration of moves. Secondly knowledge to reject certain lines of play which cannot produce the desired goal or which duplicate other search effort. Finally knowledge of positions, or position types, from which the goals can certainly be achieved. These components correspond to the better goals, holding goals and move constraints of a piece of advice.

This extra knowledge has only to preserve the correctness of the rule under consideration. We thereby introduce a hierarchy of knowledge, enabling us to demonstrate correctness locally while preserving it globally. This is very much in the spirit of structured programming. We are willing to accept a tradeoff of knowledge vs. search which provides a worst case requiring a good deal of search, for a very small percentage of the position space, if we can increment the knowledge component easily to cope with individual cases.

If the knowledge added is not correct then the strategy may not be correct. Any incorrect knowledge will however have only a local effect and can therefore be easily modified.

As mentioned earlier the exact nature of this knowledge is

dependent on the search representation. In the next section I discuss a novel representation for search in the ending, its implementation and a more precise description of the knowledge added.

6.5 Implementation

The implementation of the KPK strategy was intended as an experiment to examine new ideas with a view to incorporating them in a future chess advice language. As mentioned above this work is based on the ALI system.

The increased complexity, from the programmers point of view of this system allows much freedom of choice in implementation. The implementation of the KPK advice strategy was undertaken with the following relatively modest goals in mind:

1. The dynamic creation of pieces of advice leads us to distinguish between knowledge of <u>what</u> to do, and knowledge of <u>how</u> to do it. For a single piece of advice in an ALI advice table, the <u>what</u> to do is the achievement of the better-goal of the advice; the <u>how</u> to do it is partly specified by the holding-goals, move constraints, and depth bound. When each piece of advice is specified in advance in this manner, the user-expert is able to hand-craft the holding-goals and move-constraints for each piece of advice. This is impossible when the pieces of advice are not fully specified beforehand. In this implementation attention has been paid to the form the search algorithm must take to allow a dynamic, knowledge- based specification of the search strategy for pieces of advice.

2. Search in the endgame differs from that in the middlegame in that often each player is following a strategy relatively independent of the other. In these cases traditional search algorithms spend a great deal of time backtracking over essentially equivalent sequences of moves. This implementation explores a novel search technique which, essentially, only considers interaction between pieces. This implementation considers sets of moves rather than individual moves.

In the next section the new search technique is described, followed by a description of the search strategies available with the technique.

6.5.1 A new search technique

Chessmasters do not exhaustively search all lines of play until they find a solution, as do naive chess programs. Nor in general do they do a 1 or 2 ply search, relying on a large store of patterns to eliminate deep search (De Groot 1965). They often make deep searches along 'representative' lines of play. Although this search is only 'representative' the chessmaster will often assert that it is correct. Such a search, even with reduced material, can be far too large for a brute-force search program.

Intuitively it seems that some new mechanism is needed whereby search focusses on interaction and interference between pieces, rather than the long sequences of moves when no such

interaction takes place. In a previous paper (Bratko & Niblett 1980) we suggested a mechanism for search using square sets instead of individual squares. This eliminates the need to consider positions where no piece interaction exists. In the next section I discuss an algorithm to perform such a search in the KPK endgame. This algorithm extends very simply to any King and Pawn endgame.

6.5.2 The Algorithm

A position P = (W,B,P) will consist of 3 sets W, B and P; 1 for each piece (White king, Black king and White pawn respectively). The White pawn's set is a singleton. The interpretation of the assertion s(A,P) is now:

∀(w∈W, b∈B, p∈P) -> s(A,(w,b,p))
where (w,b,p) is a KPK position.

The pieces have in general only one move. There are three cases where the position has to be split into a set of subsidiary positions to assure that no undesired internal structure is placed on the square sets.

1. A king move is such that some of the squares to which the king might move may be illegal. In the case of the pawnset, which is a singleton, no such splitting is required. In Figure 6.5 it is WTM. Assuming no move constraints the white king can move to any square in halo(W) (the set of squares next to any square in W).

However a certain subset in halo(W) is next to B the Black king set. The set X in B is defined by X = halo(halo(W)) /\ B. The position (halo(W),B,P) is treated as the separate cases (halo(W)/halo(pi), pi, P), pi in X and (halo(W), B/X, P) thus maintaining the required absence of internal structure.

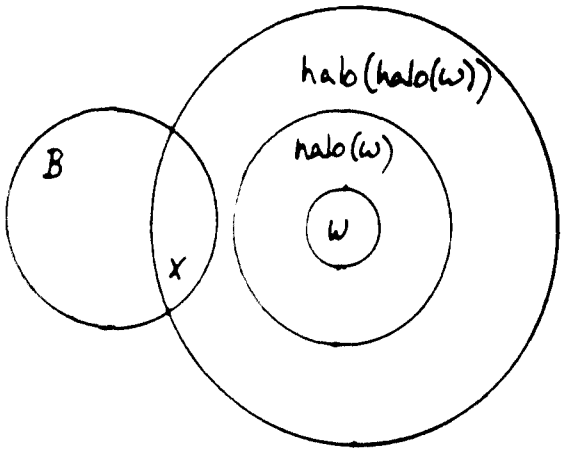


Figure 6.8

2. In some BTM position (W,B,P) where Black has a move to B', holding goals (HG) must be applied.

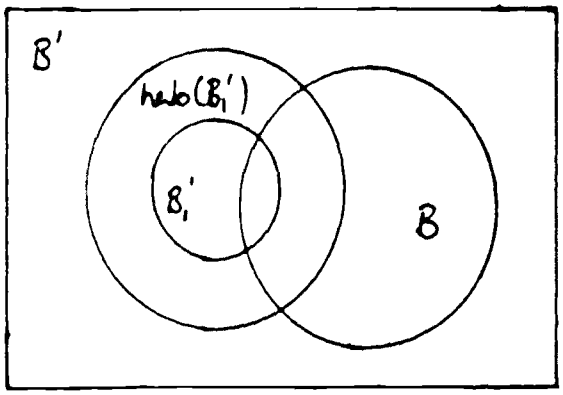


Figure 6.9

For each $b \in B'$ there is a subset $W' \subseteq W$ s.t. $\forall w \in W'$ H(w,b',P). Let the union of such b' with W' empty be B_1'. The set $B_1 = halo(B_1') \cap B$ is the set of all Black squares in the original position for which H may apply, the position is then split into cases. (W,b,P) $b \in B_1$ and $(W,B/B_1',P)$, each case being considered separately, before the application of the holding goals or Black's move.

3. The treatment of better-goals is simpler. Starting from a WTM position (W,B,P) with a White move W -> W', for each $w \in W$ there is a $B' \subseteq B$ s.t. BG(w,b), $b \in B'$. We remove B' from B to give (W', B/B',P) as the successor position.

The standard definition of a piece of advice allows for three types of search termination.

1. Search stops at nodes where the better goal is true, returning success.

2. Search stops at nodes where the holding goal is false, returning failure.

3. Search stops at nodes where there are no moves satisfying move constraints. At BTM nodes this returns success (unless stalemate), at WTM nodes failure.

Because we are dealing with sets of squares the termination conditions revolve around positions where one of the piece sets is empty.

1. If either the White king set or the White pawn set is empty search terminates with the whole Black king set passed back as the failset. The failset is defined to be the subset $Bf \subseteq B$ s.t. $\forall b \in Bf . \exists w \in W$ not(s(A,(w,b,p))), that is the failset is is the set of Black squares in which from no White king square is advice A satisfiable.

2. If the Black king set is empty the node is terminated and the failset passed back is null.

It may happen that although the failset at some node is non-empty neither of the termination conditions apply. In this case the decision as to whether to back up the failset depends on the current search strategy. This is discussed later.

6.5.3 Reduction of search

Square sets are used to reduce the amount of search. The square sets algorithm accomplishes this in three ways.

Firstly, since several moves for each piece are considered simultaneously the discovery of terminal positions is less random. A conventional program may find itself with 5 moves to consider, only 1 of which leads to a terminal position. The examination of this terminal position first requires either a fortuitous ordering of moves by the move generator or knowledge available to the search mechanism indicating the correct move. Thus the square sets algorithm will either save on time, or will need less knowledge to guide the search.

Secondly, and closely related to the above point, the fact that sets of moves are made equivalent reduces the amount of

backtracking considerably. A conventional program, unless it has a sophisticated 'causality' mechanism (Berliner 1974, Wilkins 1980) has to backtrack through every permutation of moves to guarantee failure of the advice.

Finally, the amount of search is reduced because the information returned from the search is less than with conventional programs. When a forcing tree is generated from an initial position $P = w_i b_j$ and the 'winning' move in the forcing tree is w_k to b_l we are only guaranteed that w_1 W(the advice is satisfiable from $w_k b_l$); to find a w_1 for which the advice is satisfiable requires a separate search of the forcing tree, this has not been implemented.

In the current implementation the amount of search needed (measured in nodes searched) is between 10 and 50 times less than with a conventional search containing approximately the same knowledge.

In general the square sets algorithm would seem to be very useful for general king and pawn endgames. It is less applicable to endgames with pieces such as rooks, bishops and queens where there is a very great amount of potential interaction between pieces.

6.6 Program Details and an example of play

The KPK strategy using square sets has been partially implemented in Phobos on the EDINBURGH/ICF DEC 10. A fairly complex representation for nodes is used in the current implementation. The information is detailed to allow control over the search. The top level search strategy at present implemented is shown below.

Given a plan with specified goals and move constraints:

1. set the current node to be the root
2. if the root node has non-empty failset return failure. If all moves from the root are exhausted return success
3. find an active node according to the current search strategy and set this to be the current node
4. if the current node has a non-empty failset propagate the failset back through the tree. goto 2)
5. choose a move from the current node. set the current node to be this new node. goto 4)

This strategy generates a depth-first search, checking to see whether any failsets found can be propagated back to the root node of the tree as soon as they are found.

Below is an example of play illustrating the working of the algorithm.

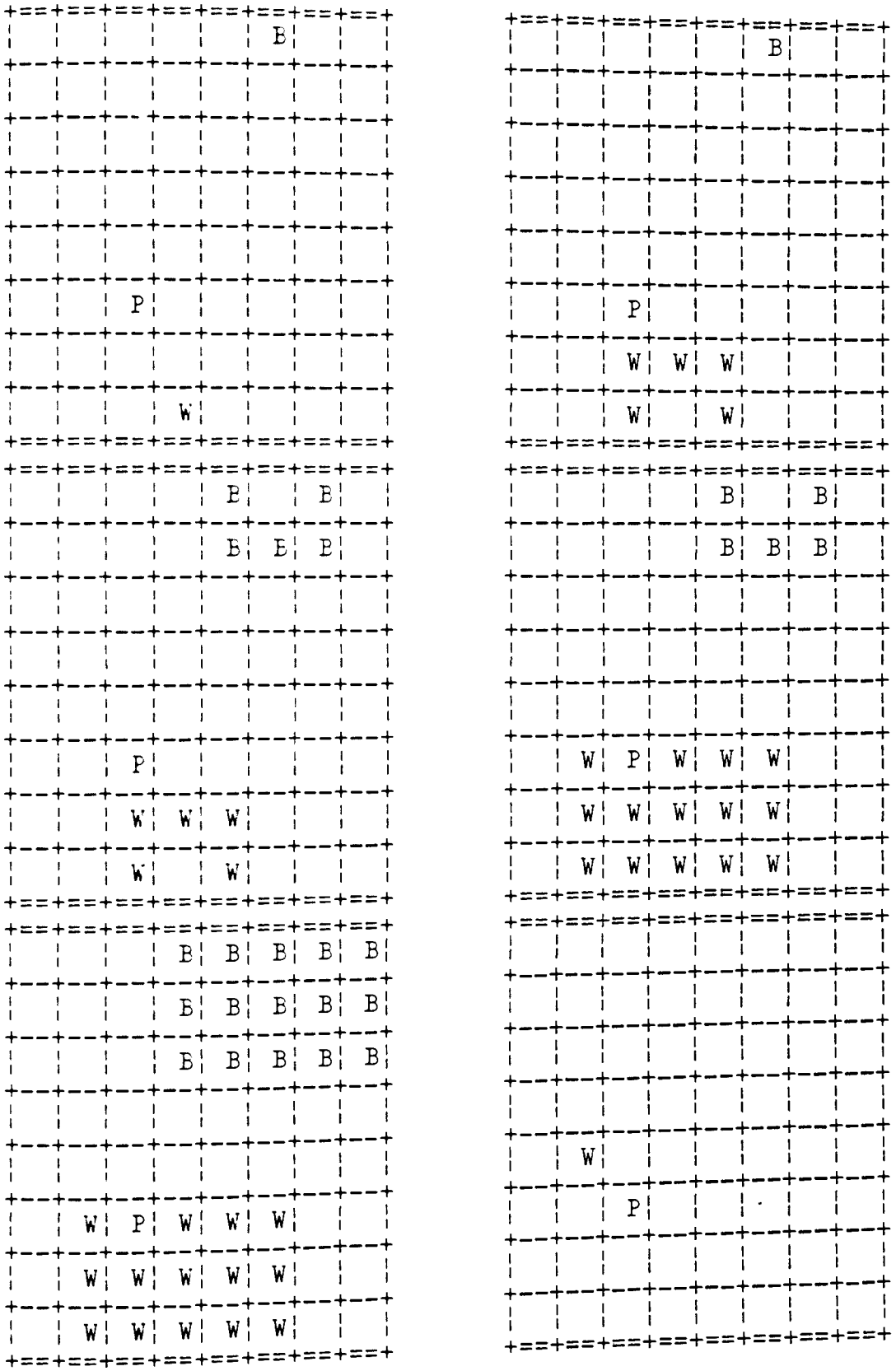
The initial position is shown in Figure 6.1b above, the program has to determine whether this is won for White. Following the KPK strategy it finds that the pawn cannot run, and then tries to determine whether mainpatt is attainable. The piece of advice being followed is:

BG: mainpatt in less than 10 ply
HG: pawnsafe and not direct opposition less than 2 ranks ahead of pawn
MG: kingmove must be able to reach mainpatt in less than 10 ply

The course of the search is shown in Figure 6.10. The first position is the initial board position. In the second position White has moved his king. The next 3 positions show moves by both sides with no interaction. In the final position we see that there are no elements in the Black king's set which do not enable White to achieve the better goal.

In this example there is no interaction between the two king sets so the search tree contains no branches. At depth 5 the Black king set disappears since White achieves the better-goal by having his king on square b4 without Black being able to seize the opposition. Due to the lack of piece interaction the holding goals and move-constraints need be only lightly constrained. A conventional tree search program finds life much harder since it considers only one possible king path at a time instead of all in parallel. Without substantial added knowledge it will spend time in unnecessary backtracking.

Figure 6.10



Search using move sets.

The search is not so simple when the goal is impossible to satisfy and there is interaction between the pieces. Figure 6.11 ilustrates a search which returns failure and involves considerable interaction between the pieces. Here extra tactical knowledge has to be added to control the search.

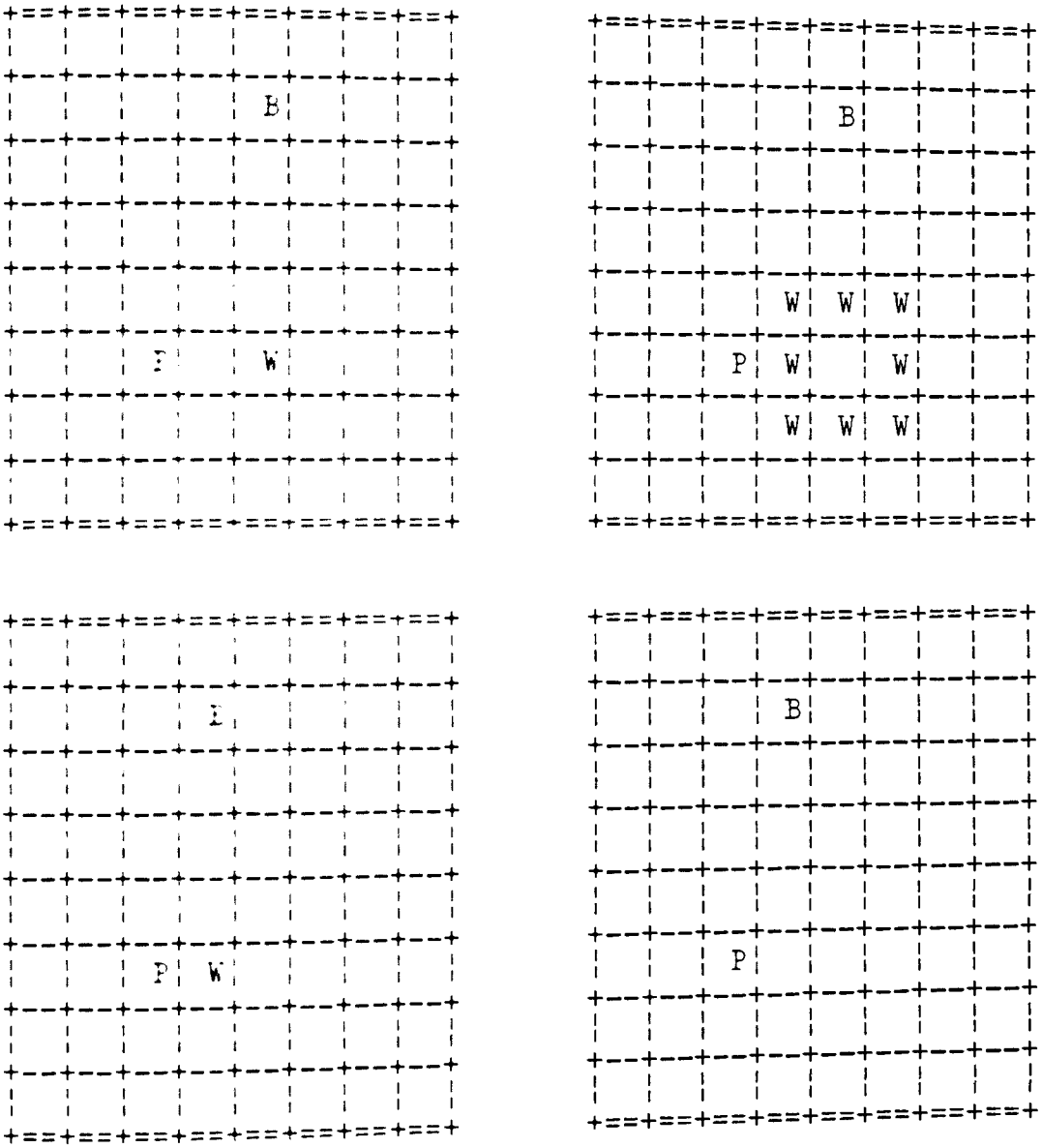
The extra knowledge added was:

- If the White king is less than 2 ranks ahead of the pawn it should not make a move which reduces its rank.
- If the White king is 2 or more squares nearer its goal than the Black king then the White king can achieve its goal.
- If the White king is 1 rank above the pawn and the Black king has opposition then White cannot achieve the goal.
- The White king should not move away from the pawn.

This extra knowledge ensures a small, constrained search. As pointed out previously this knowledge may affect correctness. In this particular case the correctness was established by database methods. This is not possible in general, and in these cases one must accept the possibility that the wrong answer may be found.

Again the first position shows the initial board position. The second position shows White's initial move. The third position shows the position after Black's first move. The other possible positions for the White king are eliminated since Black can take the opposition. The final position shows that White has no move. The Black king's set is returned as the failset.

Figure 6.11



Search with branching using move sets.

The point is that use of the square sets algorithm frees us from the need to specify the lowest level of planning for the achievement of strategic goals. By concentrating only on piece interaction we do not have to specify particular paths the pieces should follow and so prove that following these paths preserves correctness in the program. The extra tactical knowledge that must be added is simple enough and general enough to be applicable throughout the KPK domain.

Relatively little exploration of different search strategies has been done but there are one or two points to note. It is simple to determine whether a White plan succeeds <u>whatever</u> Black does, as in the first example given above. Also due to the application of the holding goals and move constraints to square sets the algorithm effectively allows a depth first search with a full width 1-ply lookahead to check for holding goal and better goal satisfaction from each node on the search path.

IMPLEMENTATION

The advice strategy has been partially implemented in PROLOG on the Edinburgh/ICF DEC 10. The examples shown above have been run and all the methods at least partially implemented. Insufficient testing has been done to determine exactly how much tactical knowledge must be added to produce acceptable run times in all cases. A straight comparison between the square sets algorithm and a normal alpha-beta search over a restricted sample

of positions shows an improvement of 50 - 100 fold in nodes searched. I have no more precise statistics for the amount of search.. The square sets algorithm seems to be very useful for king and pawn endings, where there is often very little interaction between the pieces. It is probably not useful for positions with mobile pieces such as rooks, bishops and queens.

6.7 Conclusion

The technique of search using square sets demonstrates that with an appropriate search representation the task of supplying the tactical information necessary for efficient play can be made much easier. The separation of the tactical and strategic components of the endgame allows for a straightforward proof of correctness for the advice language strategy. Even for an endgame as simple as KPK any more detailed proof involving a fully determined search strategy is unrealistic.

For a more complex endgame such as King and Rook vs. King and Knight a very large amount of tactical information is necessary. This information is considerably easier to provide by means of examples than by explicit specification. An interesting possibility is to provide the system with sufficient inductive capability to be able to learn the tactical information necessary for execution of the strategic goals. A start has been made in

this direction in (Shapiro & Niblett 82), where decision trees were induced which determine whether a KPK position satisfies any given rule in the advice strategy.

In this case the amount of 'tactical' knowledge has been taken to the limit and no search is necessary to recognise when a position is won.

7. Summary of results

Using a model based on game theory we presented a technique for the exhaustive validation of strategies which require search for their synthesis. This technique was applied to an AL1 strategy for the KPK endgame in two ways.

1. Proofs were provided for a set of lemmas sufficient to formally prove the correctness of the strategy.

2. A completely automated proof of the same strategy was provided. This second proof was less general as the techniques involved do not apply to all AL1 strategies, but only those which play in 'weak' mode or which have a depth of search of 2 ply.

Although these techniques are applicable to the more complex KNKR endgame we showed the synthesis mechanism of AL1 to be insufficient for the generation of correct strategies in a form suitable for validation. It was also argued that a correct strategy would be hard to produce at all.

An analysis of the type of synthesis technique necessary for correct generation of strategies was given and a worked example provided for the KPK endgame. It was argued that the decomposition of chess knowledge into 'tactical' and 'strategic' components via a hierarchical decomposition of strategy generation is useful.

The work described offers a useful extension of previous

methods in two ways. From the verification side the method of exhaustively analysing pieces of advice can be used to prove facts about strategies synthesised using this form of knowledge representation. We have confirmed the expressive power of this form of representation and extended its use.

On the synthesis side we have advocated the separation of differing knowledge components via a hierarchical structuring of the synthesis of playing strategies. This allows both for the use of the above-mentioned exhaustive verification techniques and for the incremental incorporation of tactical knowledge, a major problem in the synthesis and verification of chess endgame strategies.

8. REFERENCES

Arlazarov,V. & Futer,A. (1979) Computer analysis of a rook end-game. Machine Intelligence 9 361-371, (eds. J.E. Hayes, D. Michie and L.I. Mikulich). Ellis Horwood, Chichester.

Averbakh,Y. & Maizelis,I. (1974) Pawn Endings. London: Batsford.

Beal,D. & Clarke,M.R.B. (1980) The construction of economical and correct algorithms for king and pawn against king. in Advances in Computer Chess 2 1-30 (ed. M.R.B. Clarke) Edinburgh: Edinburgh University Press.

Beal,D. (1977) Discriminating wins from draws in KPK. Appendix 5 of, The construction of economical and correct algorithms for king and pawn against king. in Advances in Computer Chess 2 1-30 (ed. M.R.B. Clarke) Edinburgh: Edinburgh University Press.

Beal,D. (1982) Benefits of minimax lookahead. in Advances in Computer Chess 3 (ed. M.R.B. Clarke) Oxford: Pergamon (in press).

Berliner,H. (1974) Chess as problem solving: the development of a tactics analyser. PhD thesis, Carnegie-Mellon University.

Birmingham,J. & Kent,P. (1980) Mate at a glance. in Advances in Computer Chess 2 122-130 (ed. M.R.B. Clarke), Edinburgh: Edinburgh University Press.

Bramer,M. (1977) Representation of knowledge for chess endgames: towards a self-improving system. PhD thesis, Open University, Milton Keynes.

Bramer,M. (1980a) Correct and optimal strategies in game playing. The Computer Journal 23 (4) 347-352

Bramer,M. (1980b) An optimal algorithm for king and pawn against king using pattern knowledge. in Advances in Computer Chess 2 82-96 (ed. M.R.B. Clarke) Edinburgh: Edinburgh University Press.

Bramer,M. (1981) Machine-aided refinement of correct strategies for the endgame in chess.
Mathematics Faculty Technical Report, The Open University, Milton Keynes.

Bratko,I. (1978) Proving correctness of strategies in the AL1 assertional language.
Information Processing Letters 5 223-30.

Bratko,I. (1979) An advice program for the king-knight vs. king-rook ending.
Memorandum MIP-R-123, Machine Intelligence Research Unit, University of Edinburgh, Edinburgh.

Bratko,I., Kopec,D., & Michie,D. (1979)
Pattern-based representation of chess endgame knowledge.
Computer Journal 21 (2) 149-153.

Bratko,I. & Michie,D. (1978) AL2 design specification, Working paper, Machine Intelligence Research Unit, University of Edinburgh, Edinburgh.

Bratko,I. & Michie,D. (1980) A representation for pattern-knowledge in chess end-games.
in Advances in Computer Chess 2 31-56 (ed. M.R.B. Clarke) Edinburgh: Edinburgh University Press.

Bratko,I. & Niblett,T. (1980) Conjectures and refutations in a framework for chess endgame knowledge.
in Expert Systems in the Micro Electronic Age 83-101 (ed. D. Michie). Edinburgh: Edinburgh University Press.

Bratko,I. & Gams,M. (1982) Error analysis of the minimax principle.
in Advances in Computer Chess 3 (ed. M.R.B. Clarke) Oxford: Pergamon (in press).

Condon,J. & Thompson,K. (1982) BELLE chess hardware.
in Advances in Computer Chess 3 (ed. M.R.B. Ckarke) Oxford: Pergamon (in press).

Floyd,R. (1967) Assigning meanings to programs.
proc. symp. in Applied Math 12 19-32.

Fine,R. (1941) Basic Chess Endings.
New York: David McKay Company.

de Groot,A. (1965) Thought and Choice in Chess.
The Hague: Mouton.

Huberman,B. (1968) A program to play chess endgames. Technical report No. CS106, Computer Science Dept., Stanford University, Stanford CA.

Kopec,D. (1979) A comparison of different machine and human-oriented representations of knowledge using the king-pawn-king chess ending.
Unpublished Report, Machine Intelligence Research Unit, University of Edinburgh.

Kopec,D. & Niblett,T. (1980) How hard is the play of the king-rook-king-knight ending?
in Advances in Computer Chess 2 57-81
(ed. M.R.B. Clarke),
Edinburgh: Edinburgh University Press.

Michie,D. (1976) An advice taking system for computer chess. Computer Bulletin ser. 2 (10), pp 12-14

Mozetic,,I.(1980) User's manual for the AL1.5 system. Memorandum MIP-R-130, Machine Intelligence Research Unit, University of Edinburgh, Edinburgh.

Niblett,T. (1982) A provably correct strategy for the end-game of king and pawn vs. king.
in Machine Intelligence 10 (ed. Michie, Hayes and Pao)
Chichester: Ellis Horwood (in press).

Pitrat,J. (1980) The behavior of a chess combination program using plans.
in Advances in Computer Chess 2 110-121
(ed. M.R.B. Clarke),
Edinburgh: Edinburgh University Press.

Roycroft,A.J. (1980) Appendix 3 of,
How hard is the play of the king-rook-king-knight ending?
in Advances in Computer Chess 2 57-81 (ed. M.R.B. Clarke)
Edinburgh: Edinburgh University Press.

Sacerdoti,E. (1977) A Structure for Plans and Behavior.
New York: Elsevier North Holland.

Shannon,C. (1950) Programming a computer to play chess.
Phil. Mag. 41 256-275

Shapiro,A. & Niblett,T. (1982) Automatic induction of classification rules for a chess endgame. in Advances in Computer Chess 3 (ed. M.R.B. Clarke) Oxford: Pergamon (in press).

Strohlein,T. (1970) Untersuchungen uber combinatorische spielen. Ph.D. thesis Munich: Technische Hochschule.

Tan,S. (1972) Representation of knowledge for very simple pawn endings in chess. Research Memorandum MIP-R-98, School of Artificial Intelligence, Edinburgh University.

Wilkins,D. (1980) Using patterns and plans in chess. Artificial Intelligence 14 165-203.

von Neumann,J. & Morgenstern,O. (1944) Theory of Games and Economic Behavior, Princeton University Press, New Jersey.

Zermelo,E. (1912) An application of set theory to the theory of chess-playing. English translation of (German) paper at 5th Int. Cong. of Math. 1912, in Firbush News 6, 1976, 37-42. Machine Intelligence Research Unit, Edinburgh.

Zuidema,C. (1974) Chess, how to program the exceptions? Afdelung informatica IW 21/74. Amsterdam: Mathematisch Centrum

1 Construction of a KRKN database

The construction can be considered in two stages: initialisation and backing up. Both of these rely for their realisation on the form of the database.

To produce minimax-optimal play for KRKN it is only necessary to store the depth of win for positions won for White. This is because there are almost no positions won for Black in the position space. The database consists of two files containing information on positions, with White and Black to move respectively. These are named WHITE.DTB and BLACK.DTB.

To assign information on a given position to the appropriate

byte in these files, linear addressing functions are used. In WHITE.DTB a position (WK,WR,BK,BN) is coded as:

$$262144*BK + 4096*WR + 64*WK + WR. \quad BK \in (0...9)$$

In BLACK.DTB the same position is coded as:

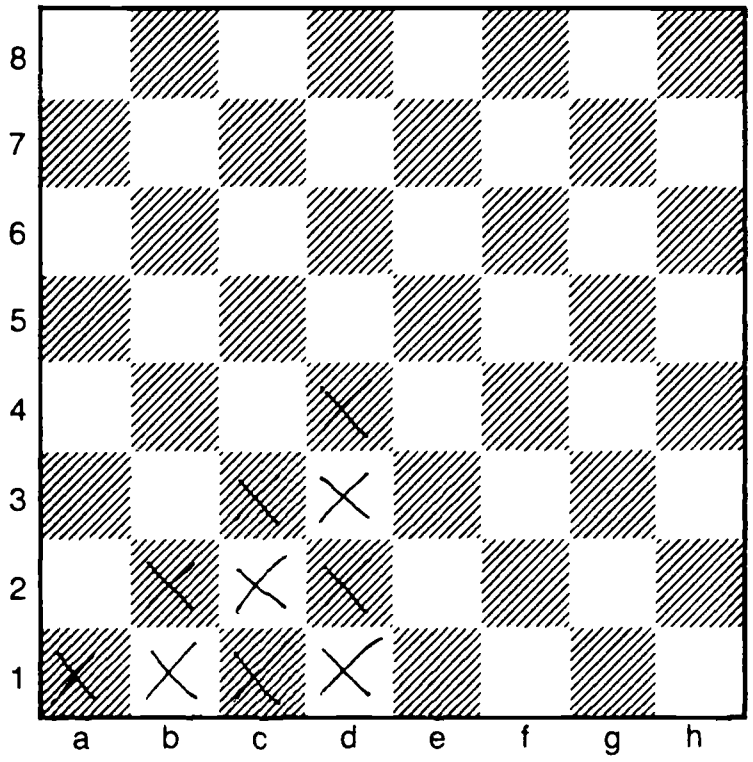
$$262144*WK + 4096*WR + 64*BK + BN. \quad WK \in (0...9)$$

The respective values of BK and WK are in the range 0..9 since by a combination of rotations and reflections the king may be considered to lie in the region shown in Figure A1.1. The original KRKN database produced by T. Strohlein did not use such a compact representation. This representation saves a lot of space but is not completely canonical. This produces complications described later.

On the file WHITE.DTB information on a position is stored as a 9-bit byte addressed by the function above. The information stored is the minimax-optimal number of moves required for White to win if the position is won. Otherwise the byte value is zero, indicating an illegal or drawn position.

The file BLACK.DTB also holds information on each position as a 9-bit byte. This information is in two parts, the first being the minimax depth of loss, Black to move. The second is a counter that holds, in the completed database, 16-(the number of moves from that position that don't lose). The first is a 5-bit byte,

Figure A1.1



Possible positions for the White king

taking rotations and reflections into account

the second a 4-bit byte. Again a value of 0 means that the position is drawn or illegal with Black to move.

Physically therefore the database is two files, each 2,621,440 9-bit bytes long. For reading and writing to and from these files, special machine-code input-output routines are used. These transfer information from disk to core. Each transfer involves 1024 36-bit words. These hold information on all positions where two pieces are fixed (i.e. WK+WR or BK+BN fixed). These words are held in two vectors in core, named WHITE and BLACK respectively.

INITIALISATION

Initialisation is in two stages. First is the setting of the counter for every Black to move position, second the marking of all immediately lost positions.

The counter is only needed for Black to move positions. P' is only marked as won for White at the given depth if every possible Black move from that position either leads to a loss at some previous depth or is illegal. Every time a position P is reached in the backing up process with P' as predecessor, the

counter for P' is incremented by one. If the counter is fifteen, instead of being incremented P' is marked as won for White at that depth.

Initialisation of the counter therefore sets it at 16-(the number of legal moves available). This is done in two stages. Firstly the number of moves that would be off the board is calculated, and secondly the moves that are illegal because thay expose the Black king to check.

Positions immediately lost are defined as positions with Black to move where

1. Black is in checkmate.
2. Black has lost the knight, cannot capture the rook and is not stalemated.

BACKING UP

At any given depth the file BLACK.DTB is searched for losses at that depth. For each of these all legal predecessors are found. If any predecessor has not been marked lost, it is marked lost at that depth. No counter is involved. After this a similar search

is made of the file WHITE.DTB. For each position marked as lost at depth N the legal predecessors are found. If these predecessors are not marked lost at a shallower depth their counters are examined. If the counter is 15 that position is marked lost at depth N+1, otherwise the counter is incremented by one.

A complication arises from the addressing function. Recall that this function is (WK)(WR)(BK)(BN) for the file BLACK.DTB and (BK)(BN)(WK)(WR) for WHITE.DTB. It is necessary to translate between these two representations when backing up. Since only one set of symmetries are considered, those for one king, there is a certain latitude for some translations. For example if WK = 1 and BK = 33 then the White king can go to 1 or 10. Backing up it is necessary to include both these possibilities to ensure that every position will have 16 successors, and therefore that the counter is accurate. Conversely if WK div 8 = WK rem 8 then only those positions where BK div 8 = BK rem 8 are backed up.

RUNNING THE PROGRAM

The program was implemented in BCPL on the Edinburgh/ICF DEC-10. Machine code was used for the critical disc I/O. Total CPU time was a little over 4 hours. However about 15 hours of time

was taken in disc I/O. This could be reduced substantially by buffering the output.

The BCPL language was chosen because of its speed, its byte manipulation facilities and because it has a very simple machine language interface.

2 Backing up Algorithms

A) STANDARD BACKUP

This is the procedure used for creating the KRK database. The algorithm is presented in a general form. It is assumed that the side whose wins we are interested in is White. Two functions are associated with each position. The first function is depth: {positions} --> {integers} which associates a depth of win with every position. This function is initially set to undef (or -1), for every position. The second function is counter: {positions} --> {integers} which is initially set to 0 for every position.

1. Set each BTM position P which is lost immediately as won for White depth 0 (depth(P) := 0).
2. For each BTM position set a counter containing the number of legal moves from that position.
3. Set i to 0
4. For each BTM position P lost at depth i (depth(P) = i), generate its legal predecesors P_1, .., P_n. If any P_j has depth(P_j) = undef then set depth(P_j) := i + 1.
5. Set i to i + 1.
6. For each WTM position P won depth i generate its legal predecessors P_1, ..., P_n. If any P_j has counter(P_j) = 1 and is not marked as won, then mark it as won depth i + 1. Otherwise decrement the counter by 1 (counter(P_j) := counter(P_j) - 1).
7. Set i to i + 1.
8. Repeat steps 4 to 7 until no new positions are marked as won. All positions from which White can force an

immediately won position will have been set with depth equal to some positive integer.

B) BACKUP WITH MOVE CONSTRAINTS

This is the backup procedure used for pieces of advice without comparison predicates, for KRK these are MATE and DIVIDE. I introduce three functions for convenience, these are: initial:{positions}->{0,1,2}, mc:{positions * positions}-> {0,1,2} and moveplayed:{positions}->list.

The function mc(pos1,pos2) has value 0 if the move pos1->pos2, pos1 being with White-to-move, cannot satisfy the move constraints for the piece of advice (note that strictly speaking each piece of advice has a separate function mc); value 1 if the move satisfies the constraint but there may be other moves with higher priority (e.g. with 'kingmove then anymove' as constraint a Rook move takes value 1); value 2 if the move satisfies the move constraint with high priority. In a similar way the function initial indicates the degree to which a position is likely to be initial. moveplayed(pos) gives a list of the moves that the Advice Table may play from a given position, it is initially nil.

1. Set all BTM positions satisfying the better goal of the piece of advice as won depth 0. Set a counter containing the number of legal moves from each BTM position. Set initial(P) = 0 for every BTM position. Set moveplayed(P) = nil for every WTM position. Set i to 0.

2. For each BTM position P lost at depth i find the legal predecessors P_1, ..., P_n. For each P_j if depth(P_j) = undef then depth(P_j) := i + 1. For each P_j if mc(P_j,P)

= initial(P_j) = 0 then moveplayed(P_j) has (P_j --> P) appended to it. For each P_j if mc(P_j,P) > initial(P_j) then moveplayed(P_j) := (P_j --> P) and initial(P_j) := mc(P_j,P).

3. i := i + 1.

4. Backing up from WTM positions is the same as for the standard backing up procedure.

5. When either steps 2 to 4 produce no new won positions or the depth bound of the piece of advice is reached then stop.

C) BACKUP WITH COMPARISON PREDICATES

This is the most complex backing up procedure for KRK. Three extra functions are introduced, back:{positions}->{integers}, cp:{positions}->{integers}, and used{positions}->{true,false}.

The function used indicates whether a position has occurred more than once in the backing up procedure, back is used to minimax the values of the comparison predicate and cp gives the value of the comparison predicate in any particular position. We assume that the comparison predicate is to be maximised by White, that is the better goal is of the form: 'value at root of forcing tree must be less that value achieved by execution of forcing tree'.

1. Set all BTM positions as 'won' for White depth 0. Set a counter containing the number of legal moves available for every BTM position. For every BTM position P set back(P) = cp(P). For every WTM position set back(P) to be 1 plus the max value the comparison predicate can take. Set i to 0. For every WTM position P set used(P) to false.

2. For every BTM position P, won at depth i, find the legal predecessors P_1, ..., P_n. For each P_j if depth(P_j) = undef then depth(P_j) := i + 1 and back(P_j) := back(P). Else for each P_j if back(P_j) < back(P) then back(P_j) := back(P) and used(P_j) := true.

3. i := i + 1.

4. For each WTM position P won at depth i find the legal predecessors P_1, ...,P_n. For each P_j if counter(P_j) > 1

and used(P) = false then counter(P_j) := counter(P_j) - 1. If counter(P_j)=1 and depth(P_j) = undef then depth(P_j) := i + 1. If back(P) < back(P_j) then back(P_j) := back(P).

5. i := i + 1.

6. Continue steps 2 to 4 until the depth bound for the piece of advice is reached or no new positions are marked as won. All WTM positions P where back(P) < cp(P) are positions from which the piece of advice is satisfiable.

IMPLEMENTATION

The proof system has been implemented in BCPL on the Edinburgh DEC 10. The reasons for using BCPL are documented in Appendix 1 on the construction of a King and Rook vs King and Knight database.

The program is split into several modules:

1. Move generation. A series of fast routines using various lookup tables to produce predecessor positions in the KRK domain.

2. Input-output routines: These handle the input and output of large files to and from disc store. These are written in machine code for efficiency.

3. Transpose routines: These transpose the large files used from White-to-move representation back to Black-to-move and vice versa.

4. Backup routines: These do the actual work of backing-up. Since the algorithms for backing-up are quite different there are two sets of these.

The file structure is in principle very similar to that of my KRKN database. There are two main files WHITE.KRK and BLACK.KRK holding information on positions with White and Black-to-move respectively. A third file COPY.KRK is used to transpose from one representation to another.

As a check on all these routines, and since it is of great interest for statistical purposes, a database for the KRK game was generated. The correctness of this has been checked by comparing the figures for number of positions lost at each depth with those given by Clarke.

3 Correctness proof for the KPK strategy

To prove the correctness of the KPK strategy I shall prove the following:

1. A WTM position is won if and only if it matches the right hand side of one of rules 2 - 8 in the advice strategy
2. If a WTM position matches the right hand side of one of these rules then the corresponding plan on the left hand side wins for White

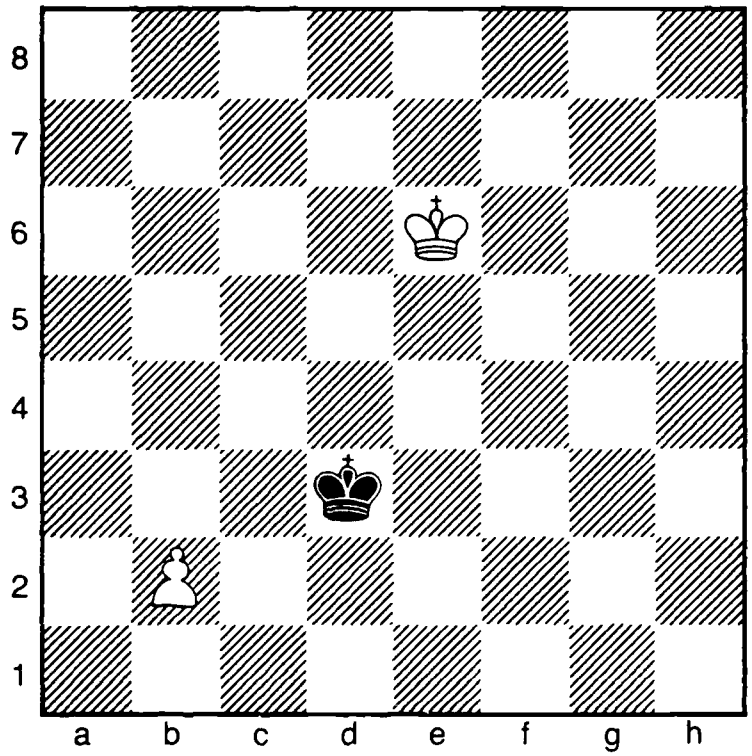
A proof of 2) can be used to provide a proof of the sufficiency condition in 1). The proof of necessity in 1) is extremely simple. This simplicity arises from the use of the better goal canrun\/mainpatt in rule 8. The table is still correct if we substitute the simpler better goal mainpatt. This has been established via a lengthy case analysis.

The disjunction reduces search in many cases. Consider Figure A3.1. It takes White only one move (Kd5) to achieve a position where the pawn can run. To achieve mainpatt requires three moves and results in much less efficient play. The proof below relies on the fact that the goal canrun is also sufficient on its own.

Theorem

If a WTM position is won then then one of rules 2 - 8

Figure A3.1



White to move

After 1. Kd5 ... the pawn can run.

White needs 5 ply to achieve mainpatt

in the advice strategy of section 6.3 matches.

PROOF

A KPK position is won for White iff White can promote the pawn safely. Thus from every won position there is a forcing tree T for the piece of advice:

WBG: pawn on 8th rank and promoted
WHG: pawnsafe and not(stalemate)
WMC: nil
BMC: nil

Every branch in T must contain a position Q where White is to move, the pawn is on the 7th rank and can advance safely. The predicate canrun is true in Q. The tree T' with terminal nodes at each such Q is a forcing tree for the piece of advice of rule 8.

This proof although simple does not indicate any bounds on search. Such bounds will be considered later.

Theorem

If a WTM position matches one of rules 2 - 8 then

the corresponding plan wins for White

PROOF

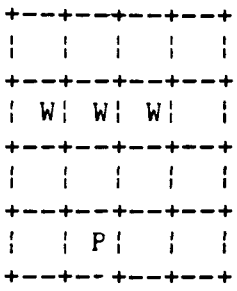
The proof proceeds by considering the various rules that can be matched. The proof hangs on the following lemma which ensures that rules 6 and 7 can only be applied a finite number of times.

Lemma

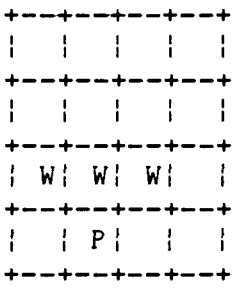
From any WTM position P where rank(pawn,P) < 7, the pawn is not on the rooks's file, and where mainpatt holds, it is possible to reach a position where mainpatt holds with the pawn further advanced.

Proof

We split mainpatt into two subpatterns. In one the White king is 2 ranks ahead of the pawn, in the other 1.



Pattern 1



Pattern 2

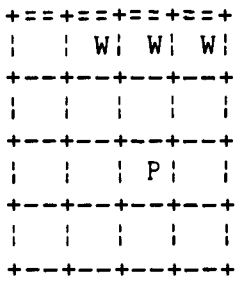
The definition of Patterns 1 and 2 are as follows,

- Pattern1(P) iff wtm(P) and not rookfile(p,P) and rank(p,P) = rank(w,P)-2 and abs(file(p,P)-file(w,P)) < 2 and not stalemate(P).

- Pattern2(p,P) iff wtm(P) and not rookfile(p,P) and rank(p,P) = rank(w,P)-1 and abs(file(p,P)-file(w,P)) < 2 and not stalemate(P) and the 2 kings are not in direct opposition.

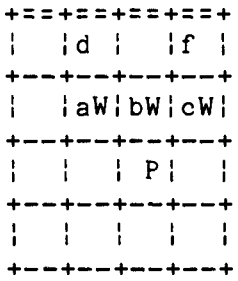
We proceed by cases. The proof is given diagrammatically.

<u>rank(pawn,P</u> = 6)



Advance the pawn

Pattern 2 advanced



If W is on b move to a or c. With W on a or c advance P, B any, W to d or f, B any.
Pattern 2 advanced.

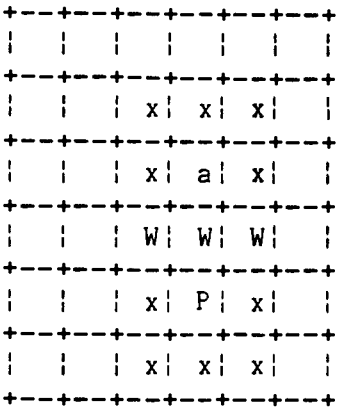
For all other pawn ranks we show that from pattern 2, either pattern 1 is achievable or pattern 2 with the pawn advanced is achievable. We also show that from pattern 1 either pattern 2 is achievable with the pawn advanced, or pattern 1 is achievable with the pawn advanced.

Lemma

Given pattern 2 we can either achieve pattern 1,

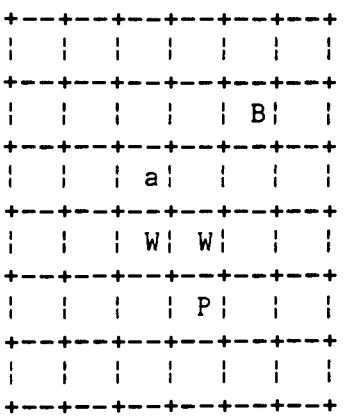
or pattern 2 with the pawn advanced.

Case a)



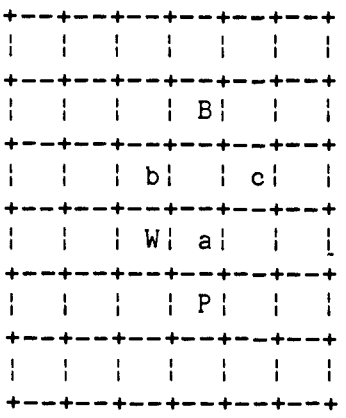
B on any but X, W to a - pattern 1.

Case b)



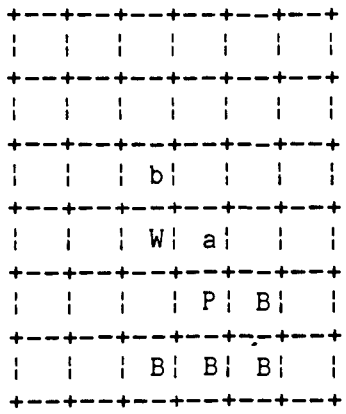
W to a - pattern 1. (a similar case obtains by reflection in the vertical axis of symmetry)

Case c)



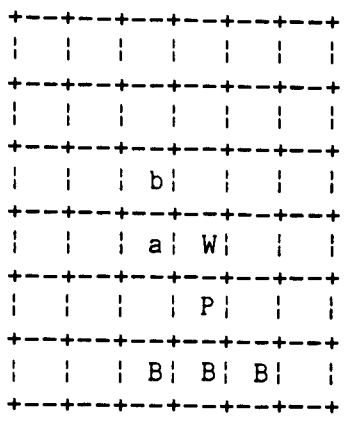
W to a, B any, W to b or c - pattern 1.
(a similar case again obtains by symmetry)

Case d)



P to a, B any, W to b - pattern 2 advanced.
(a similar case obtains by symmetry)

Case e)



W to a, B any, P advance, W to b - pattern 2 advanced.

Case f)

For any case other than a) to e) and their symmetrical variants White advances the king to get Pattern 1.

Lemma

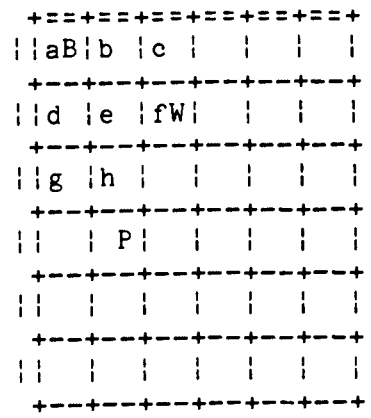
Given pattern 1 it is possible to achieve one or other of the two patterns with the pawn advanced.

(Note: this lemma shows that mainplan works)

case a)

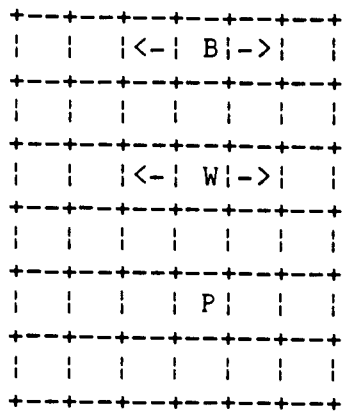
The general case is taken to be where White can advance the pawn to achieve Pattern 2 with the pawn advanced. The cases considered below are the exceptions to this rule.

case b)



W to h, B to b, W to g. Now if:
1) B to c, W to d followed by P to h - Pattern 2 advanced.
2) B to a, P to h to e followed by W to a - Pattern 2 advanced.
3) B to f, W to d followed by P to h - Pattern 2 advanced.

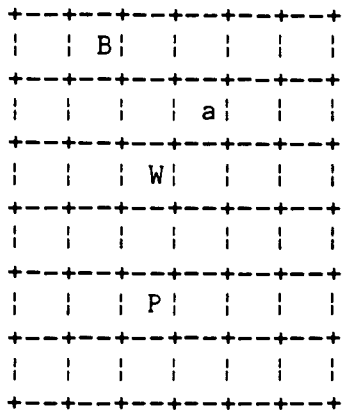
case c)



(W and B in direct opposition)

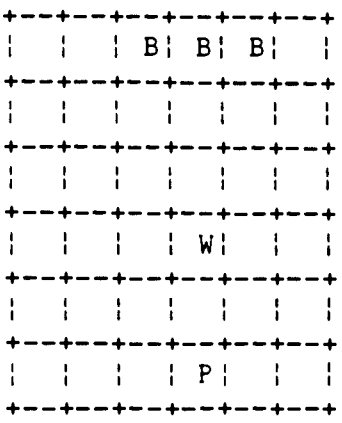
advance P - pattern 2 advanced.

case d)



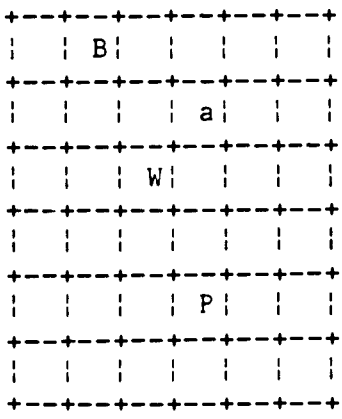
W to a, B any, advance P - pattern 1 advanced.
(a similar obtains by symmetry)

case e)



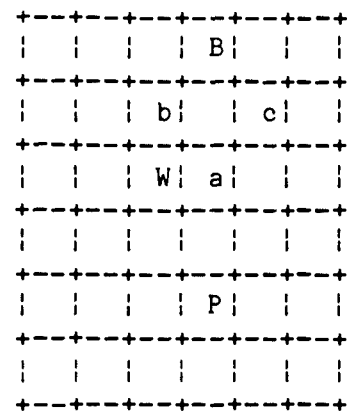
advance W, B any, advance P - Pattern 1 advanced.

case f)



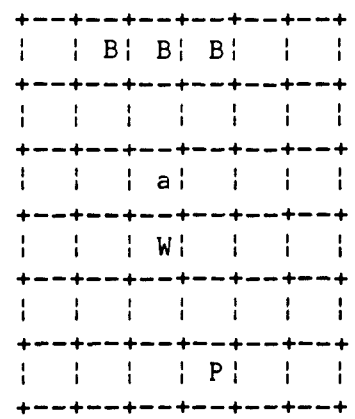
W to a, B any, P advances - pattern 1 advanced.
(a similar case obtains by symmetry)

case g)



W to a, B any, W to b or c, B any,
P advances - pattern 1 advanced.
(a similar pattern obtains by symmetry)

case i)



W to a, B any, P advances - pattern 1 advanced
(a similar case obtains by symmetry)

This completes the proof of the lemma. We are now prepared to prove the main theorem.

Again we proceed by cases

1) Rule 2 is matched

If the pawn can run it can reach the eighth rank without Black being able to capture it. White will always win by promoting the pawn to a rook, unless the pattern of Figure A3.2 holds. In this case White wins by any king move followed by promotion of the pawn.

2) Rule 3 is matched

Either canrun or rookpatt is attainable. 1) above deals with canrun. If rookpatt is attainable then the following strategy wins for White.

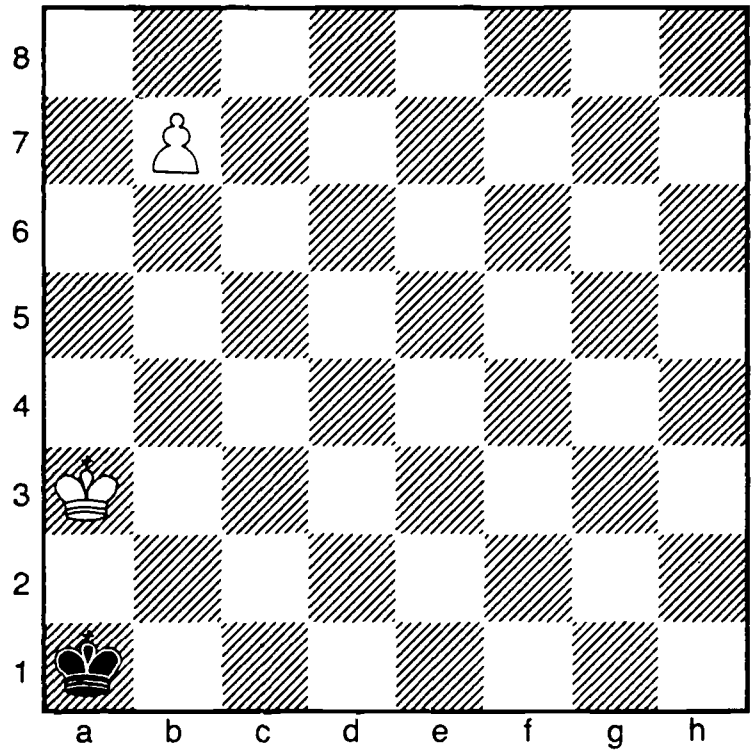
Rookplan

If the Black king moves towards the eighth rank, ahead of the White king then advance the White king 1 rank unless it is on the seventh rank. If the Black king moves from the eighth rank to the 7th, move the White king to the seventh rank. Otherwise advance the pawn.

The proof that rookplan always wins from ROOKPATT is straightforward, and not given here. It is necessary to show that

1. White can only make a finite number of moves before moving the pawn
2. Black cannot capture the pawn if the Black king moves 2 ranks below the White king at any time

Figure A3.2

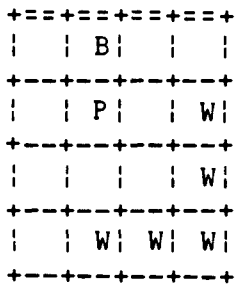


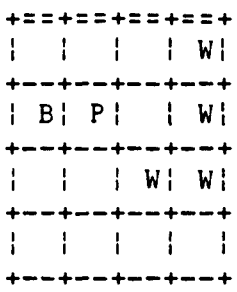
White to move

If White queens or rooks the pawn immediately stalemate results. White must make a king move before advancing the pawn.

3) Rule 4 is matched

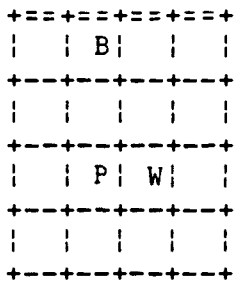
One of the situations below holds, after execution of rule 4 rule 2 will be matched.





4) Rule 5 is matched

The pattern below is matched. After execution of the rule rule 4 will be matched.



5) Rule 6, 7 or 8 is matched.

If rule 6 is matched then after execution of the rule either pattern-rank6 or mainpatt holds. If the former then we are done

since rule 5 will match. If mainpatt holds then either one of rules 2 - 5 will be applied or rule 7. Application of rule 7 advances the pawn and maintains mainpatt by the above lemma. This can only occur a finite number of times. When the pawn is on the 7th rank rule 2 or rule 4 will match. If rule 8 matches then it can only be applied once. Figure A3.3 shows the possible sequences of rule application for any won position.

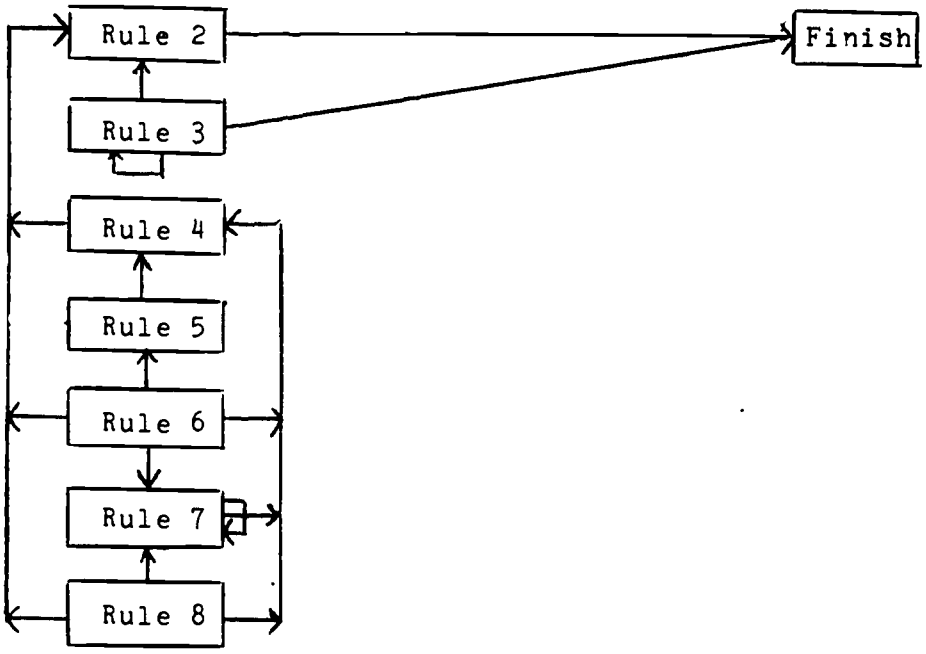
This completes the proof of correctness. It is also possible to show that a weaker version of rule 8 is sufficient. This would be useful if the evaluation of canrun were expensive. The replacement rule is,

```
wins_with(white,P,<<mainpatt,pawnsafe,nil>;apply(Rule)>)
<-- s((mainpatt:pawnsafe:nil:nil),P).
```

The two lemmas below bear the burden of the proof. Consider a won WTM position P. There exists a forcing tree T from P for the piece of advice with better goal canrun \/ mainpatt. We need to show that there is a forcing tree T' for the better goal mainpatt.

It is possible to transform T into T'. It is sufficient to show that at any terminal node of T where the pawn can run mainpatt is achievable and that at any WTM terminal node where the pawn can run either mainpatt is achievable or White could have chosen a move where it is.

Figure A3.3



Possible sequences of rule application

The lemmas are sufficient to prove this.

Lemma 1

If in a WTM position P the pawn cannot run and after a king move to Q the pawn can run, then mainpatt is attainable from Q.

Note: There exist BTM positions where the disjunctive goal canrun \/ mainpatt is achievable, but not MAINPATT alone.

Proof

This lemma can be proven by considering cases. Whenever the White king can make a move which results in the pawn being able to run where it could not before, we can show that the White king is on Black's path to the pawn. The proof is tedious and a database method is simpler.

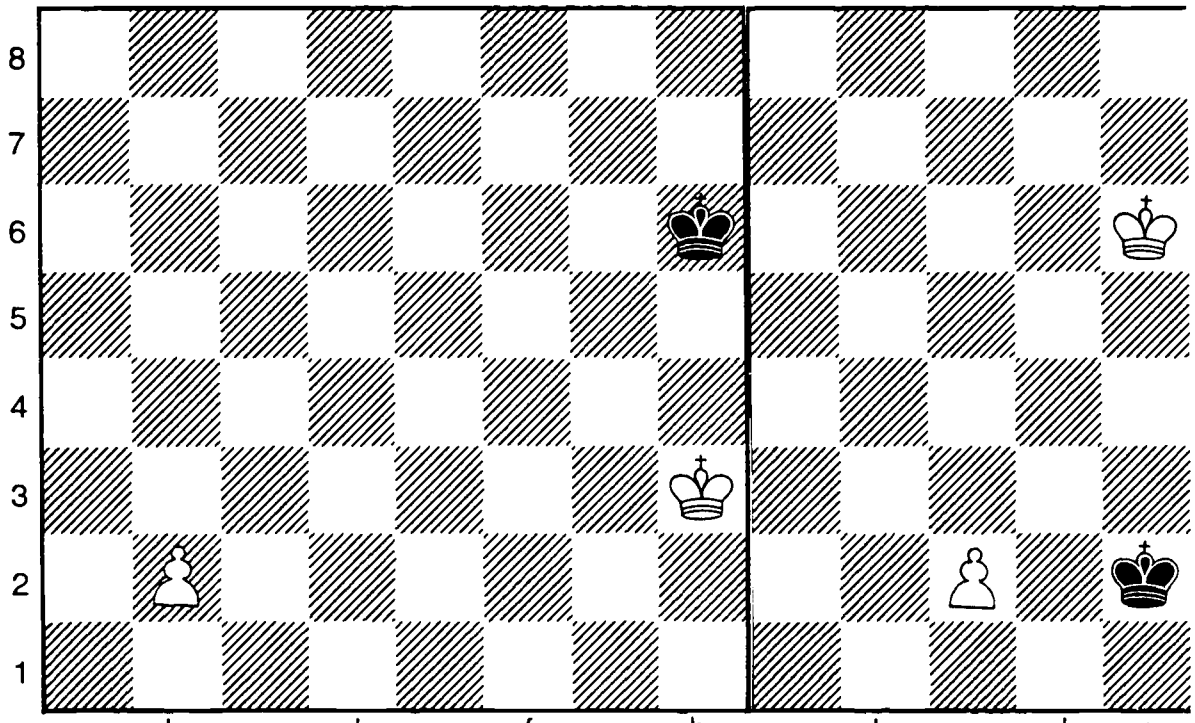
Lemma 2

Given a BTM position P, won for White, which has a successor P' where the pawn can run and a successor P'' where mainpatt is achievable then from any WTM predecessor of P mainpatt is achievable.

I cannot see how to produce a proof by cases for this lemma. Instead all BTM positions satisfying the conditions of the above

lemma were generated by database techniques. There are 22; they fall into 2 classes illustrated in Figure A3.4. The first class is of positions where if the Black king seizes the opposition the pawn can run and if the Black king chases after the pawn the White king can achieve the opposition or move 2 ranks ahead of the pawn. The second class is of positions where if the Black king moves next to the pawn the pawn can run, otherwise mainpatt is achievable.

Figure A3.4



4 Description of the KNKR tables

Figure A4.1 shows the composition of the 12 pieces of advice in Bratko's original table. The extra pieces of advice in the second table are described below (these are numbers 13, 14, 15 and 16). Bratko also modified pieces of advice nos. 10 and 11, these are also given below. I then give a description of the predicates used in the 2 tables. 'Us' and 'Them' can be taken as White and Black respectively.

THE EXTRA PIECES OF ADVICE

Advice 13 APPROACH2

```
UBG: UBGTACT6
TBG: nil
UHG: not TRSAFE or (not MATE and not ONLOST and ONSAFE2P
                    and not EDGELOSS and not ONTRAPPD)
THG: TRDEAD or (ONSAFE and not ONCORN and (DEPTHEQ2 or
                not NSAFETYL and OKONNDLE))
MC:  BESTM1 then ... then BESTM10
DEPTH: 6
```

Advice 14 APPROACH3

```
UBG: DEPTHEQ4 or UBGTACT6
TBG: nil
UHG: not TRSAFE or (not MATE and not ONLOST and ONSAFE2P
                    and not EDGELOSS and not ONTRAPPD)
THG: TRDEAD or (ONSAFE and not ONCORN and not NSAFETYL)
MC: BESTM1 then ... then BESTM10
DEPTH: 4
```

Advice 11 SURVIVE1

```
UBG: nil
TBG: nil
```

Figure A4.1

	Advice No.												
	1	2	3	4	5	6	7	8	9	10	11	12	
OKONNDLT	-	-	-	-	-	-	-	-	-	Y	-	-	UTM BG
TRDEAD	Y	-	-	-	-	-	-	-	-	-	-	-	
NOT MATE	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	
NOT ONLOST	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	UTM HG
ONSAFE2P	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	
TRDEADorCHECK	Y	-	-	-	-	-	-	-	-	-	-	-	
NOT ONOKATT	-	Y	Y	Y	Y	-	-	-	-	-	-	-	
OKEDNGE	-	Y	Y	-	-	-	-	-	-	-	-	-	
OKCONDGE	-	Y	-	Y	Y	Y	-	-	-	-	-	-	
OKONDLE2	-	Y	Y	Y	Y	-	Y	-	-	-	-	-	TTM HG
KINGSCLOSE	-	Y	Y	Y	-	-	-	-	-	-	-	-	
OKONDLE3	-	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	
TKONDGT1	-	Y	-	-	Y	Y	-	-	-	-	-	-	
OKONDEQ3	-	-	-	-	-	-	-	-	Y	-	-	-	
CHECK THEN ANY	x	x	x	x	x	x	x						
NMOVE									x				
NMOVE THEN ANY										x			
MAX. DEPTH OF SEARCH	3	4	4	4	4	4	4	4	1	4	4	2	

```
UHG: not TRSAFE or (not MATE and not ONLOST and ONSAFE
                    and not EDGELOSS)
THG: nil
MC: BESTM1 then ... then BESTM10 then ANYMOVE
DEPTH: 4
```

Advice 16 EDGECASE

```
UBG: nil
TBG: CHECK and not OKONNDLE
UHG: nil
THG: nil
MC: NMOVE
DEPTH: 2
```

Advice 15 APPROACH0

```
UBG: UBGTACT6
TBG: nil
UHG: not TRSAFE or (not MATE and not ONLOST and ONSAFE2P
                    and not EDGELOSS and not ONTRAPPD)
THG: TRDEAD or (ONSAFE and OKONNDLE and not NSAFETYL
                and not ONCORN)
MC: BESTM1 then ... then BESTM10
DEPTH: 6
```

Advice 10 APPROACH1

```
UBG: UBGTACT6
TBG: nil
UHG: not TRSAFE or (not MATE and not ONLOST and ONSAFE2P
                    and not EDGELOSS and not ONTRAPPD)
MC: BESTM1 then ... BESTM10
DEPTH: 6
```

PREDICATES USED

OK, ON, TK and TR stand for the Black king, Black knight, White king and White rook respectively. CURPOS is the current board position to which the advice is applied, NEWPOS is any position in the forcing tree except for CURPOS. Function dist(p_1,p_2,pos) gives the distance in king moves between pieces p_1 and p_2 in position pos. ON mobility is the number of safe squares to which ON can move. Function mdist(p_1,p_2,pos) gives the

Manhattan distance between p_1 and p_2 in pos. The function nsafety(p) for position p is defined as:

$$nsafety(p) = 5 \times v(On,p)^2 + v(s,p)^2$$

where S is the set of squares attainable by ON in 1 move from position p and,

$$v(s,p) = dist(TK,s,p) - dist(OK,s,p) + 6.$$

UBGTACT6: At depth 6 in search tree: true.
At depth 4 if OK is not in check and
neither TK or TR attacks ON then true
else false.At depth 2 if not TRSAFE or
STALEMATE then true else if OK is in check or
TK or TR attacks ON or ON mobility has decreased
after the last them-move or OK is on the edge and TK is
at dist 2 from OK or OK in corner and
mdist(OK,TK,NEWPOS)<4 then false
else true.

TRSAFE TR cannot be captured immediately.

EDGELOSS Pattern of Figure A4.2

ONTRAPPD Pattern of Figure A4.3

NSAFETYL nsafety(NEWPOS) < nsafety(CURPOS).

ONCORN ON in the corner.

DEPTHEQ<N> Depth of NEWPOS in the search tree is N.

BESTM1, ..., BESTM10 Move constraint predicates such that BESTMi is true for the i-th highest ranked move by the nsafety measure.

OKEDGE Our king is on the edge of the board.

OKONSEP Our king and knight are more than 4 king moves apart.

CORNCASE Special pattern of Figure 5.2.

EDGECASE Special pattern of Figure 5.3.

OKONNDLT Our king and knight are less far separated

than they were at the root node.

TRSAFE Their rook cannot be immediately captured.

TRDEAD Their rook has been captured.

NOT MATE It is not checkmate.

NOT ONLOST Our knight has not been captured.

ONSAFE2P Our knight is safe from a few 2-ply deep pins and forks.

ONOKATT Our knight attacks their king.

OKEDNDGE Our king is closer to the edge of the board than it was in the root position.

OKCONDGE Our king is closer to the corner of the board than it was in the root position.

OKONDLE2 Our king and knight are less than 2 king moves apart.

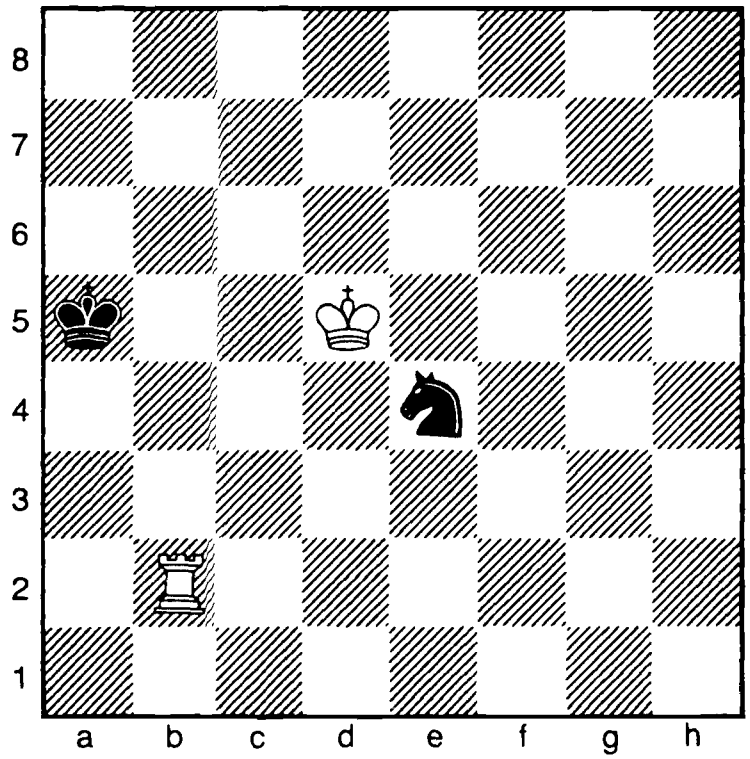
KINGSCLOSE The two kings are less than 3 king moves apart.

OKONDLE3 Our king and knight are less than 4 king moves apart.

TKONDGT1 Their king and our knight are more than 1 king move apart.

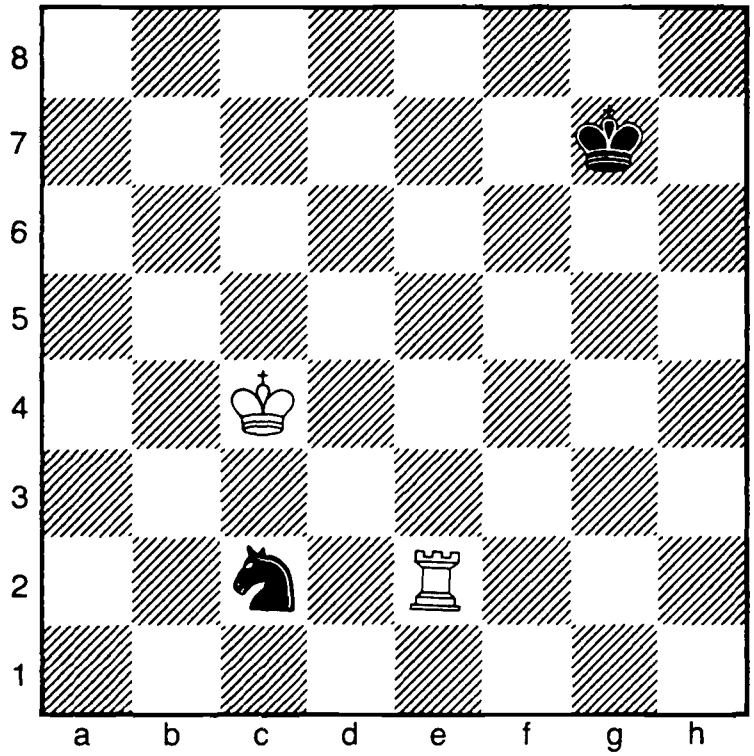
OKONDEQ3 Our king and knight are 3 king moves apart.

Figure A4.2



The EDGELOSS position(BTM)

Figure A4.3



An ONTRAPPD position