INFORMATICA 2/1982

AN EFFICIENT IMPLEMENTATION OF ADVICE LANGUAGE 2

IVAN BRATKO[1,2], IGOR KONONENKO[1], IGOR MOZETIČ[2]

1 FACULTY OF ELECTRICAL ENGINEERING, E. KARDELJ UNIVERSITY, TRŽAŠKA 25, LJUBLJANA
2 JOŽEF STEFAN INSTITUTE, JAMOVA 39 LJUBLJANA

UDK: 519.682.7

Advice Languages were developed to facilitate the implementation of knowledge-based solutions to problems that are combinatorial in nature. This paper reports on an efficient implementation of AL2 (in Fortran), and experiments with its application to chess endgames. In particular, a difficult ending with king and knight vs. king and rook was implemented in AL2 following the structure of an older advice program which was previously written in AL1. The new implementation of this ending, in AL2, is about two orders of magnitude faster than the previous AL1 program.

UČINKOVITA IMPLEMENTACIJA JEZIKA NASVETOV AL2. Jeziki nasvetov olajšujejo reševanje kombinatoričnih problemov s pomočjo aktivne baze znanja. V članku je opisana učinkovita implementacija AL2 in eksperimenti z njegovo uporabo v šahovskih končnicah. Na podlagi prejšnjega programa napisanega v AL1, je bila v AL2 implementirana težka končnica kralja in skakača proti kralju in trdnjavi. Implementacija v AL2 je od prvotne, programirane v AL1, hitrejša za dva reda velikosti.

1. INTRODUCTION

One useful way of looking at problems is: given an initial problem situation, S, find a sequence of actions which transforms S into a final situation, F, such that F satisfies a given goal condition G. The actions can be chosen from a predefined repertoire of permissible operations on problem situations. This general concept is reflected in various formalisms for representing problems, widely used in artificial intelligence, such as state-space, AND/OR graphs, or game-trees (e.g. Nilsson 1980). Examples of practical problems which naturally fit these formalisms are: combinatorial optimization, computer-aided design, automatic programming, failure detection and diagnosis, game-playing, etc.

When solving problems within this paradigm, the domain-specific knowledge has to be used for practical reasons in order to reduce the combinatorial complexity of the search process. One way of expressing knowledge for streamlining the problem-solving process is to specify subgoals which are relevant to the given goal. This can be done in the framework of production rules in the form popularly used in artificial intelligence (e.g . Waterman and Hayes-Roth 1978). A series of rule-based systems, called Advice Languages, has been based on this idea (AL1: Bratko and Michie 1980a; 1980b; AL2: Mozetic 1980; AL3: Bratko 1982a; 1982b). In this paper we describe an implementation, in Fortran, of AL2. AL1 and AL3 were implemented (elswhere) in AI languages POP-2 and Prolog respectively.

In experiments with the Advice Languages, the game of chess has been mainly used as an experimental domain. The adversary nature of games introduces complications, but the idea of breaking the whole problem into subgoals is still viable and rather straightforward. Consider, for example, the ending with king and rook vs. king (KRK for short). A useful piece of advice for this ending would be: in order to mate, first squeeze the opponent's king against the edge, and then try to mate. The whole problem of mating has been reduced into the (easier) subgoals: force the opponent's king to the edge, and, when on the edge, carry out the final phase for mate.

This is, basically, the form in which the knowledge about problems is communicated to the problem-solver via Advice Language 2.

Two nontrivial endgames have been implemented in AL1 and AL3: king and knight vs. king and rook (KNKR) in AL1 (Bratko and Michie 1980b), and king and pawn vs. king and pawn in AL3 (Bratko 1982a; 1982b). As shown by Kopec and Niblett (1980), the difficulty of perfect play in KNKR is beyond the capabilities of chess masters. The performance of an advice program in AL1 for KNKR was, if not perfect, at least comparable to that of human masters.

One trouble with AL1 was its slowness, due to an inefficient implementation in POP-2. To play a move in a nontrivial KNKR position, it took the system typically a few minutes of CPU on a DEC KI-10 processor. We translated the AL1 program for KNKR into AL2 and used it as a benchmark for testing our present implementation of AL2. The present implementation typically uses between a second and a few seconds to play a move in a difficult KNKR position on a DEC KL-10 processor. The pace of play of a few seconds per move is attained by AL2 when searching about 1000 positions in the game-tree. Under regular chess tournament conditions, the player is allowed about three minutes per move. So under tournament conditions, the program would be allowed to search correspondingly larger portions of the game-tree. With, say, 50 times more time per move, the system would be able search some 50 thousand of positions per move actually played across the board. This figure is comparable to the speed of some (not fastest !) tournament chess programs.

The rest of the paper describes the details of AL2 and its implementation, and some experimental results.

2. AL2 DESIGN

The overall structure of the AL2 system is depicted in Fig. 1. The main modules of the system are :

(1) the knowledge base
Knowledge is represented in advice-tables and pieces-of-advice. Each advice-table is specialised as to deal with certain problem-subdomain. According to the current problem-situation it chooses an apropriate advice-list. Advice-list is an ordered list of pieces-of-advice. A piece-of-advice suggests what goal should be achieved next while preserving some other condition. If this goal can be achieved in a given problem-situation then we say that the piece-of-advice is satisfiable.

(2) the search module
The search module takes an advice-list from the knowledge base and tries to satisfy a piece-of-advice in the list, sequentially attempting the pieces one after another, until it succeeds. To satisfy a piece-of-advice is to find a specific subtree of the game-tree called forcing-tree. Forcing-tree may be interpreted as a strategy that guarantees us the achievement of the goals prescribed in the first satisfiable piece-of-advice from the advice-list.

(3) the interactive interface
The interface facilitates the communication between the user-player and the system. It accepts a forcing-tree from search module and interprets it as our strategy for playing our moves. Besides this it can show useful statistics and enable trace during the search for a forcing-tree.

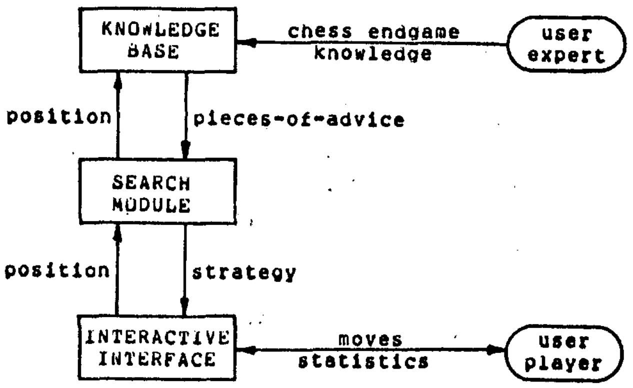


Fig. 1: The basic structure of the AL2 system.

It should be noted that the basic structure of AL2 as described above is almost identical to that of the AL1 system (implemented in POP-2 at the University of Illinois, outlined also in Bratko and Michie 1980).

In the following paragraphs we describe the modules of AL2 in more details.

2.1. The knowledge base

Advice-table is a set of rules of the form

if precondition then advice-list

If more then one precondition is satisfied then simply the first rule is chosen.

A piece-of-advice is a five-tuple

(HG, BG, UMC, TMC, MAXD)

where HG and BG are predicates on positions called holding-goal and better-goal respectively, UMC and TMC are predicates on moves, called move-constraints for us and for them respectively and MAXD is a depth-bound of a forcing-tree. By "us" we mean the side to move in a given position, either white or black, and by "them" the opponent of "us".

Move-constraints can besides a mere selection of a subset of legal moves prescribe an ordering on the moves that are selected.

We say that a piece-of-advice is satisfiable in a given position if and only if :

(1) the "us" side can force the achievement of the better-goal in less then MAXD plies, while
(2) during the play toward the better-goal, the holding-goal is never violated, and
(3) the "us" side always chooses its moves only from the set of moves that satisfy UMC, and
(4) the "them" side's choice of moves is limited only to the moves that satisfy TMC.

Goals may also be expressed by the satisfiability of some other piece-of-advice, as follows :

Better-goal (or holding-goal) is achieved in a position if the other piece-of-advice is satisfiable in the same position.

Take, as an example, the king and pawn vs. king ending. A piece-of-advice for the stronger side that suggests simply "push the pawn to the queening square" can be defined as :

(pawn_safe, pawn_queened, pawn_moves,
king_macro_move_to_queening_square, 9_plies)

If we take for example the position in Fig. 2, it is obvious that this piece-of-advice fails, because the black king can capture the white pawn.

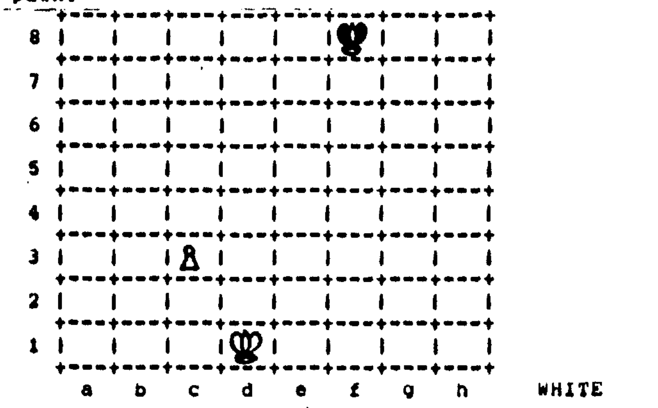


Fig. 2: An example of nontrivial position for white to move and win. After the natural move K d2 black can draw by K e7. The only correct move is K c2 !.

2.2. The search module

Control structure chooses which piece-of-advice from advice-list to apply next. Then it tries to satisfy it by simple depth-first search. If and only if a piece-of-advice is satisfiable in a given position there exists a forcing-tree and the search module finds it.

Forcing-tree is a subtree of the game-tree rooted in a given position, such that :

(1) every node, except the root-node, satisfies HG,
(2) every nonterminal node satisfies not BG,
(3) every terminal node satisfies BG,
(4) there is exactly one move from every nonterminal us-to-move node; that move must satisfy UMC,
(5) there are all legal moves from every them-to-move nonterminal node that satisfy TMC,
(6) the lenght of the longest path must not exceed the depth-bound MAXD; the root-node is supposed to be at depth 0.

We say that in a position, from which there is no legal "us" move that satisfies UMC, the better-goal can not be achieved. But if there is no legal "them" move that satisfies TMC, the position is a terminal node of a forcing-tree.

In Fig. 3 there is an example of forcing-tree generated for the position in Fig. 2. Here, a piece-of-advice for white that suggests how to get the king in front of the pawn is satisfied.

```
WK c2   BK e7   WK b3   BK d6   WK b4   BK c6  /
                                        BK d5  /
                        BK e6   WK c4   BK d6  /
                        BK d7   WK b4   BK d6  /
                                        BK c6  /
        BK f7   WK d3   BK e7   WK c4   BK d6  /
                        BK e6   WK c4   BK d6  /
        BK e8   WK b3   BK e7   WK c4   BK d6  /
                        BK d7   WK b4   BK d6  /
                                        BK c6  /
```

Fig. 3: The forcing-tree for white for the position in Fig. 2. On any black's move white has an answer that brings his king in front of the pawn, so that black cannot eventually prevent the queening.

```
 ADVICE  NODES    1   2   3   4   5   6   7   8
-----------------------------------------------
   1       19     1   5   1   5   1   4   1   1
   2       58     3   9   6  15  10  15   0   0
 NODES =   77         (CPU time = 0.18 sec)
```

Fig. 4: Statistic typed during game-tree search for white in the position from Fig. 2. The first piece-of-advice "push_pawn" fails in 8 plies, but the second "bring_king_in_front_of_pawn" succeeds.

2.3. The interactive interface

The interface can interpret some dozen commands from the user, to start various functions of the system. The search module triggers the searching for a forcing-tree on the user-player request. This one is then interpreted as our strategy and executed by interactive interface in across-the-board play.

The user-expert may specify the system to play in the "strong-mode" or in the "weak-mode". In the first case the strategy is executed up to a terminal node in the forcing-tree. On the other hand if the "weak-mode" is chosen it means that there is no point in following the current strategy up to its end, and a new forcing-tree is generated immediately on the next move.

3. IMPLEMENTATION

A Fortran implementation of the AL2 system was developed on a PDP-11/34 at "Jozef Stefan" Institute, Ljubljana. Later it was installed on a DEC-10 at the University of Edinburgh and at the E. Kardelj University at Ljubljana. The development of the AL2 programming package which contains about 3000 Fortran commands required about a half a year work of one of us.

Fortran is not very convenient language for such a kind of programming task, but at the moment when the work was initiated no other high-level programming language was available. On the other hand the Fortran implementation has turned out to be much more efficient then the known implementations in others, more powerful languages (POP-2, Prolog). For example, the current Fortan implementation on DEC-10 is about 100 times faster and even on PDP-11/34 about 10 times faster then POP-2 implementation of the AL1 system. The reason is only partly due to the 4 times faster processor in the first case. The efficiency is measured in the number of positions generated per second during the game-tree search. The AL2 system examines between 500 and 1000 positions per second, depending on the complexity of predicates in pieces-of-advice.

4. EXPERIMENTS AND RESULTS

Two major experiments conducted with AL2 were concerned with the implementation of two endgames: king and pawn vs. king (Mozetic 1980), and king and knight vs. king and rook (KNKR). Here we present the second experiment, the more complex test of AL2: a KNKR advice program. Our AL2 program for KNKR is, in fact, a translation of an advice program for KNKR in AL1 (Bratko and Michie 1980b).

The following is a summary of basic ideas underlying the KNKR advice program. The program plays for the weaker side, i.e. the knight's side. If a KNKR position is theoretically won for the rook's side, the win is based on two motifs: first, mating threats, and second, capture of the knight. The latter is possible either after a short, forced combination (e.g. based on pinning the knight), or after a long-term strategic plan which consists of separating the knight and its friendly king, and subsequently surrounding the knight. If neither of the two basic motifs can be exploited then the position is theoretically a draw. The advice program for the knight's side is based on negating the attacker's goals. The corresponding broad advice is as follows:

1. Avoid mate: keep the king as far from the edge as possible; keep the king as far from the corner as possible.

2. Preserve the knight: keep the knight as near the friendly king as possible and far from the enemy king.

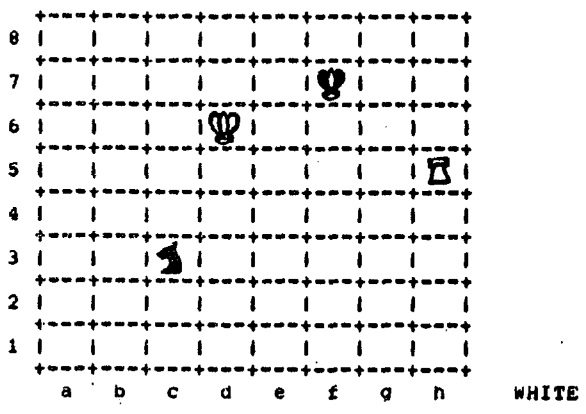
The details of this broad strategy are incorporated into an advice table comprised of 5 columns (which corresponds to 5 production rules) and 16 pieces-of-advice.

As our KNKR advice program is a rather straightforward translation of the original program in AL1 we here present only examples of its behaviour, specially with respect to its efficiency.

In the following examples we give the CPU time the system spent for playing the move, and for some positions statistics on the number of positions searched (for separate pieces-of-advice tried before the move was found, showing also these numbers over the depth of positions in the game-tree; depths correspond to columns labeled 1-6).

Example 1:

This is a game between the advice program and an international chess master. The starting position of the game is in the following diagram. This position is won for white, as known from databases of complete KNKR positions. White can force the capture of the knight against best defence in 24 moves.



1. K e5 , N a4 ! (1.88 sec)

(The best defence, rather hard to find.) Statistics on search follows.)

ADVICE	NODES	1	2	3	4	5	6
1	14	14	0	0	0	0	0
15	172	14	25	30	31	21	51
14	172	14	25	30	31	21	51
13	423	7	54	47	86	30	199

NODES = 781

2. R h7+, K e8 (2.54 sec, 1078 nodes)

3. K d6 , N b6 (2.46 sec, 1074 nodes)

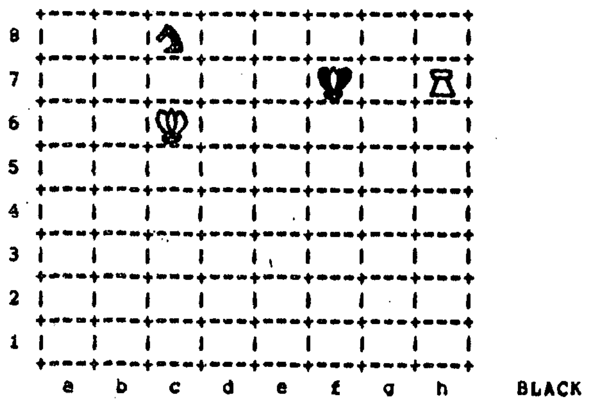
4. R h8+!, K f7 (0.98 sec, 389 nodes)

5. R h4 , N c8+ (1.98 sec, 880 nodes)

6. K d7 , N b6+ (0.54 sec, 218 nodes)

7. K c6 , N c8 (1.44 sec, 594 nodes)

8. R h7+, ...



8. ... , K e6? (5.78 sec)

ADVICE	NODES	1	2	3	4	5	6
1	6	6	0	0	0	0	0
15	828	6	65	18	100	71	568
14	828	6	65	18	100	71	568
13	896	6	105	33	113	71	568
12	93	1	18	7	67	0	0

NODES = 2651

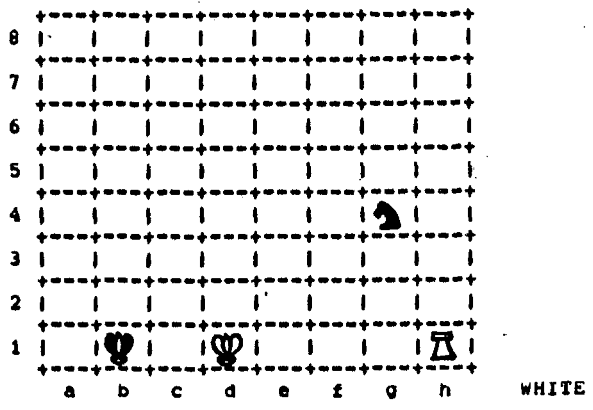
Up to this point both the program and the master played optimally. This is the first mistake in this game which considerably shortens the resistance of the weaker side. Correct was K f6!.

9. R h6+, K e5 (0.90 sec, 464 nodes)

Play was stopped here as the white's win is clear by: 10. K d7, N a7 11. R a6, N b5 12. R a5 etc.

Example 2:

In the following game against another chess master the program again defended a lost position. This time, after a master's mistake the program saved the game.



1. R h4!, N f6?! (2.32 sec)

ADVICE	NODES	1	2	3	4	5	6
1	39	9	6	24	0	0	0
15	980	1	17	13	186	91	672

NODES = 1019

The move N f6 was good, but not optimal. Best was N e5.

2. K d2 , K b2 (1.00 sec, 417 nodes)

3. R d4?

After this mistake the position is theoretically drawn. Correct was K d3.

3. .... , K b3 (1.06 sec, 434 nodes)

4. K d3 , N e8 (0.74 sec, 287 nodes)

5. R d7 , K b4 (0.88 sec, 388 nodes)

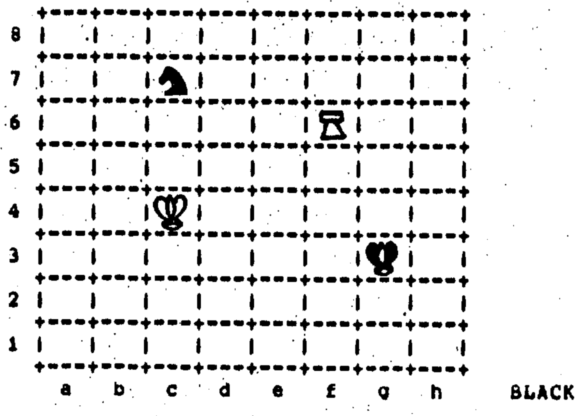
6. K d4 , K b5 (1.02 sec, 448 nodes)

7. R e7 , N d6 (1.66 sec, 717 nodes)

8. K d5 , N c8 (0.64 sec, 241 nodes)

At this point draw was agreed. Position is obviously drawn.

Example 3:

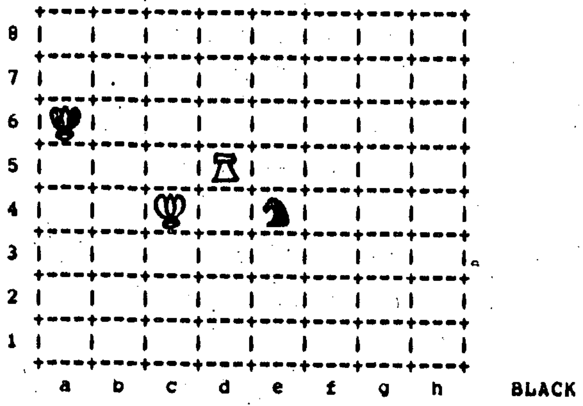


Program plays: K h4 (3.60 sec)

ADVICE	NODES	1	2	3	4	5	6
1	11	11	0	0	0	0	0
15	196	11	20	15	38	16	96
14	349	11	42	28	84	21	163
13	956	5	62	36	201	48	604
NODES =	1512						

The move played by the program K h4 is the only, study-like drawing move. The natural move K g4 would lose after R c6, N d8, R e6.

Example 4:

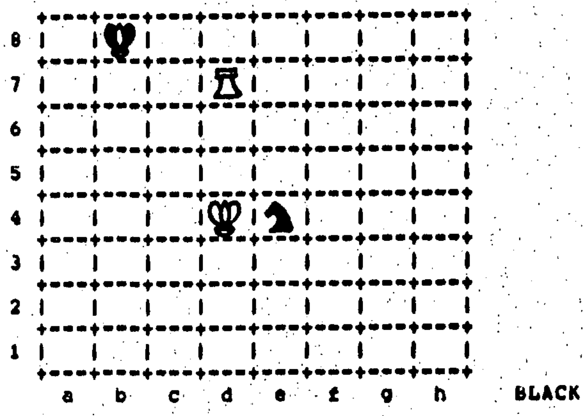


Program plays: K b7 (2.06 sec)

ADVICE	NODES	1	2	3	4	5	6
1	13	11	2	0	0	0	0
15	706	4	37	29	142	49	445
NODES =	719						

The move K b7 is again surprizing, but only correct. The intuitive move K b6 loses after R d4 forcing the knight far from its own king.

Example 5:

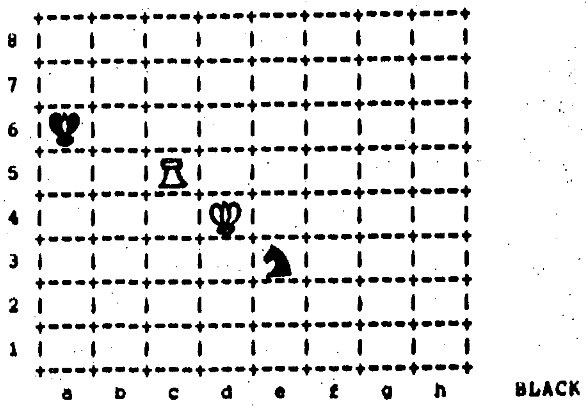


The program plays the only drawing move:

K c8 (2.24 sec)

ADVICE	NODES	1	2	3	4	5	6
1	10	10	0	0	0	0	0
15	897	5	32	18	152	58	632
NODES =	907						

Example 6:



The program finds the pretty and only correct:

K b6 (0.86 sec)

ADVICE	NODES	1	2	3	4	5	6
1	13	11	2	0	0	0	0
15	271	3	19	27	34	26	162
NODES =	284						

5. CONCLUSION

The implementation of AL2, presented in this paper, is by about two orders of magnitude faster than the previous implementation of AL1. AL2 is also currently the only implementation of Advice Languages whose legal move generator is complete. Others are limited to certain subgames only.

AL2 as a language is very similar to AL1, it only introduces small but useful improvements. The major improvement over AL1 thus lies in AL2's efficiency. On the other hand, AL3 is, compared to AL1 and AL2, as a nonprocedural programming language, much more flexible and general. However, its present inefficiency makes it a rather impractical tool for solving realistic problems. One attractive possibility is to combine the AL3's linguistic power and AL2's efficiency. Thus the high level reasoning would be done in the domain of AL3. The result of this would be 'plans', or pieces-of-advice, which would be passed to AL2 for concrete checking by search.

REFERENCES

1. Bratko, I. (1982a) Knowledge-based problem-solving in AL3. Machine Intelligence 10 (eds. D.Michie, J.H.Pao) Ellis Horwood and Wiley.

2. Bratko, I. (1982b) Advice and planning in chess endgames. NATO Symposium on Artificial and Human Intelligence, Lyon, Oct. 1981. To appear in Artificial and Human Thinking (eds. A.Elithorn, R.Banerji, in preparation).

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

4. Bratko, I., Michie, D. (1980b) An advice program for a complex chess programming task. The Computer Journal, Vol. 23, pp. 353-359.

5. Kopec, D., Niblett, T. (1980) How difficult is the play in the KNKR ending? Advances in Computer Chess 2 (ed. M.R.B.Clarke) Edinburgh University Press.

6. Nilsson, N.J. (1980) Principles of Artificial Intelligence, Tioga Publishing Co.

7. Mozetic, I. (1980) User's manual for the AL2 system, An AL2 strategy for king and pawn vs. king. Research Memorandum MIP-R-130, Machine Intelligence Research Unit, University of Edinburgh.

8. Waterman, D.A., Hayes-Roth, F. (1978) Pattern-directed Inference Systems. New York: Academic Press.