3

Knowledge-based problem-solving in AL3

I. Bratko
Institute Jozef Stefan and Faculty of Electrical Engineering
University of Ljubljana

1. INTRODUCTION

AL3 (Advice Language 3) is a problem-solving system whose structure facilitates the implementation of knowledge for a chosen problem-domain in terms of plans for solving problems, 'pieces-of-advice', patterns, motifs, etc. AL3 is a successor of AL1 and AL1.5 (Michie 1976, Bratko & Michie 1980a, 1980b, Mozetic 1979). Experiments in which AL1 was applied to chess endgames established that it is a powerful tool for representing search heuristics and problem-solving strategies. The power of AL1 lies mainly in the use of a fundamental concept of AL1: piece-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 (e.g. a given chess position) then we say that the piece-of-advice is 'satisfiable' in that position. In this way AL1 makes it possible to break the whole problem of achieving an ultimate goal into a sequence of subproblems, each of them consisting of achievement of a subgoal prescribed by some piece-of-advice. The control structure which chooses what piece-of-advice to apply next consists of a set of 'advice-tables', each of them being specialized in a certain problem-subdomain. Each advice-table is a set of rules of the form

if precondition *then* advice-list

If more than one rule-precondition is satisfied then simply the first rule is chosen. Advice-list is an ordered list of pieces-of-advice. Advice-list is interpreted so that the first satisfiable piece-of-advice in the list is executed. The satisfiability is checked by simple depth-first search.

This comparatively simple control structure has several advantages: simplicity, neatness of solutions, susceptibility to formal proofs of correctness of strategies. However, its disadvantage is that it is difficult to implement problem-solving strategies which make extensive use of higher-order concepts, such as plans, and which also do 'meta-level' reasoning about plans and pieces-of-advice themselves.

It is sometimes desirable that the system be able to create a new piece-of-advice, or in a case of its failure, modify it according to the cause of the failure.

AL 1.5 removed some minor defects of AL1. One improvement was to allow recursive calls of pieces-of-advice within a piece-of-advice. But the basic control structure of AL1 was preserved.

AL3 is an attempt at facilitating the use of higher-order concepts by providing a more flexible control structure over the basic mechanisms of AL1. Experiments with AL3, described in this paper, were implemented in PROLOG (Pereira, Pereira & Warren 1978). The problem-domain used in these experiments is a chess ending: king and pawn *vs.* king and pawn with passed pawns. Examples of using AL3 in another chess ending are described in Bratko & Niblett (1979). Although these experiments demonstrate how AL3 can be used for knowledge-based problem-solving using higher-order concepts, at this stage they should not be considered as completed. Many questions need further investigation, such as: in what ways, in general, can different plans be merged for achieving a desired combined effect? Examples of related research, also using plans for chess problem-solving, are Tan (1977), Pitrat (1977), and Wilkins (1979).

2. EXAMPLE: SOLVING A CHESS STUDY

As an illustration of the way AL3 uses problem-solving knowledge, consider the chess endgame study in Fig. 1. The Black pawn on h5 is threatening to run down to h1 and promote into a queen. White can try to stop this threat with his king, but the king on h8 is too far to catch the Black pawn. Another idea for White

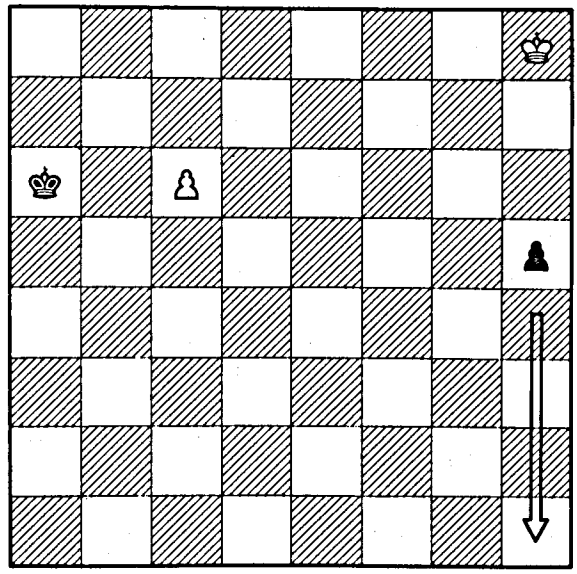


Fig. 1 – A study by Reti. White to move: can White draw? The Black pawn is threatening to run to square h1 and promote into a queen as indicated by the arrow.

is to queen his own pawn. But if the White pawn moves ahead, the Black king can easily move closer to stop the pawn and capture it if necessary. The position looks lost for White. However, there is a surprising plan for White which preserves the draw.

The following is a simplified trace of AL3's behaviour when solving the problem of Fig. 1. The problem is stated by an initial query 'Can White draw?' and the following hypothesis, *H*0, is investigated:

*H*0: White can draw?

This causes another hypothesis to be generated

*H*1: Black can win?

and a logical relation between both hypotheses is provided:

$H0 \Leftrightarrow \text{not}(H1)$

That is: *H*0 is logically equivalent to *not H*1. This logical relation is called a 'fact' about our problem.

Now the system may consider the hypothesis *H*1, and one *method* for solving problems in the system's knowledge-base suggests a plan for Black: push Black pawn toward h1 while preserving a *holding-goal* "Black pawn alive and not White pawn successfully promoted" until a *better-goal* is achieved: "Black pawn promoted". Call this plan *BPQ* (Black pawn queens). This and subsequent plans are illustrated in Fig. 2. A hypothesis about plan *BPQ* is now generated

*H*2: Plan *BPQ* succeeds?

together with the fact

$H2 \Rightarrow H1$

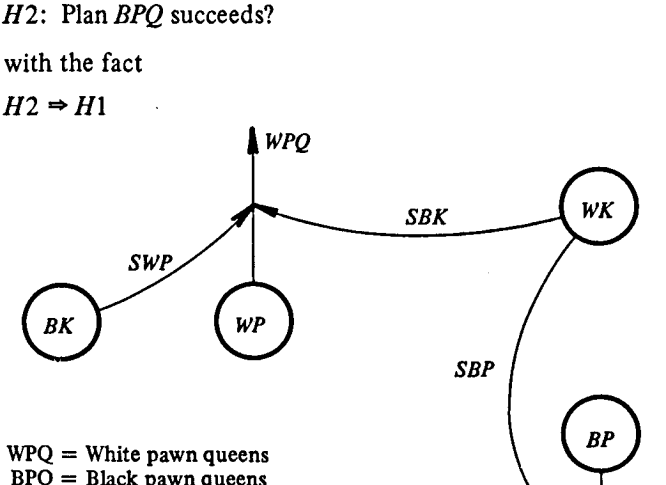


Fig. 2 – Illustration of plans for White and Black in the position of Fig. 1.

A method which 'knows' how to refute plans now proposes: try to refute plan *BPQ* by either destroying its holding-goal or its better-goal. Thus two refutation plans are proposed for White:

1. Plan *SBP* (Stop Black pawn): move White king toward the Black pawn path.
2. Plan *WPQ* (White pawn queens): promote White pawn by pushing it toward square c8.

Two corresponding hypotheses and a logical relation are:

*H*3: *SBP* refutes *BPQ*?
*H*4: *WPQ* refutes *BPQ*?
$H3 \vee H4 \Rightarrow \text{not}(H2)$

There is a lemma in the knowledge-base which recognizes on the basis of distances among pieces that *H*3 is false. But another lemma, about pawn races, establishes that *H*4 is true. This gives, using logical relations among hypotheses: *H*2 is false. Thus the simple plan for Black, *BPQ*, (push Black pawn and queen) does not succeed.

One method in the knowledge-base, considering this failure and the cause for the failure, proposes a refinement of plan *BPQ*, obtaining plan *BPQ*1. The skeleton of *BPQ*1 is *BPQ*, but in the meantime Black king has, if necessary, to stop the White pawn by moving toward the White pawn path. Now either of the White's counter plans *WPQ* and *SBP* refutes the plan *BPQ*1.

The repertoire of simple ideas for White is now exhausted, but more complicated ideas can still be tried. First, plan *WPQ* is refined, obtaining plan *WPQ*1. The skeleton of *WPQ*1 is *WPQ* refined by the idea of bringing the White king toward the White pawn path in order to prevent Black's plan *SWP* (stop White pawn). It turns out that *WPQ*1 also does not refute *BPQ*1. But there is one more idea for White: a disjunctive combination of plans *WPQ*1 and *SBP*. The plan, based on this idea, *WPQ*1 *or SBP*, does refute Black's plan *BPQ*1. The solution that saves the White position is finally: White king moves diagonally from h8 to f6 or e5, all the time pursuing the idea of the 'or' combination of plans *WPQ*1 and *SBP*. The diagonal White king moves serve *both* plans. Then, depending on Black's reactions, one of the component plans refutes Black's *BPQ*1, ensuring the draw.

3. OVERVIEW OF AL3

The overall structure of the AL3 system is shown in Fig. 3. The main modules of the system are:

(1) a knowledge-base which contains *methods* that 'know' how to solve particular problems, and *lemmas* (or theorems) about the problem-domain that, hopefully, can be applied during the problem-solving process;

(2) a current-knowledge-pool (CKP) containing the already known *facts* and *hypotheses* about the problem being solved, and other objects that are relevant to the problem;
(3) the *control module* which decides what method, or lemma, to activate next.

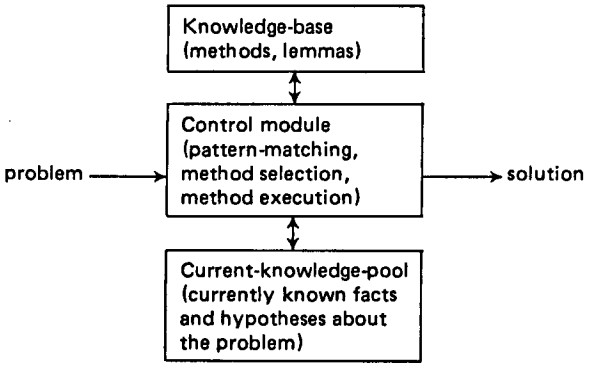


Fig. 3 – The AL3 System.

3.1 Knowledge-base

Methods in the knowledge-base are, in the terminology of Waterman & Hayes-Roth (1978), pattern-directed executional modules. They can be thought of as specialized subroutines for solving particular kinds of subproblems, or for providing a suggestion about how to solve a (sub)problem. For example, one method for the problem-domain of the king and pawn *vs.* king and pawn chess ending says: If one side (say White) is planning to promote his pawn, then a counter-plan for Black is: stop the advancing White pawn by bringing the Black king in front of the pawn. Together with this, a hypothesis is generated that this Black's plan refutes White's plan, and the following fact is provided: If the hypothesis is true then White's plan fails. The necessary precondition for this method to be executed is the existence of a hypothesis that White can promote his pawn.

Each method is defined by (1) its precondition, (2) a procedure for its execution, and (3) its characteristics. The characteristics include, for example, an estimate of how difficult is the method to execute; that is: how much computational resource will be spent on the execution of the method.

Preconditions are predicates on the current-knowledge-pool (CKP). They are implemented so that they do not only return the truth value. In the case that the precondition is satisfied, the 'context' which satisfies the precondition is also returned as a by-product of testing for the precondition. Context is simply a part of CKP.

When a method is executed, the context is used as input for the method's procedure. The results of the execution can be: new hypotheses, new facts, new plans for solving the problem, or other objects. These results are then used to update the CKP.

A special class of methods is called *lemmas* to indicate that by them we implement theorems about the problem-domain. Formally there is no distinction between methods and lemmas. The only difference is that methods may generate new hypotheses whereby lemmas generate only facts.

Facts are propositional calculus formulae made of hypothesis names. Thus for example, the fact that a hypothesis H is true can be represented by a formula:

$$H$$

The fact that if hypothesis $H1$ is true then $H2$ is false can be represented by

$$H1 \Rightarrow \text{not}(H2)$$

3.2 Current-knowledge-pool

CKP contains:

- hypotheses about the problem including the user's definition of the problem which is to be solved,
- user's query, called a 'target', which is to be proved or disproved, together with the currently known facts about the problem,
- plans, pieces-of-advice, and other objects that are in any respect relevant to the problem-solving task and have thus been generated so far during the problem-solving process.

3.3 Control module and executional cycle of AL3

The control module supervises the problem-solving process which consists of a sequence of executional cycles. To carry out each executional cycle the control module does the following: it analyses the current target and checks if enough facts about the problem are already known to imply an answer. If not, then the control module matches the preconditions of the methods and lemmas against the CKP to find a set of methods and lemmas applicable to CKP. This set is called the *conflict set*. A method or a lemma in the conflict set is then selected on the basis of a cost-benefit criterion. The selected method will, hopefully, produce new facts so as to most economically further the problem-solving process.

A PROLOG code for the top level of operation of AL3, including the main executional cycle, is given in Fig. 4. Notational conventions are those of the Edinburgh implementation of PROLOG (Pereira, Pereira, & Warren 1978). For solving a problem, the target is initialized by the user's query, and an upper limit on computational resources that may be spent on solving this problem is specified. The PROLOG procedure

```
solve(Target, Resources, Answer, Explanation)
```

produces an answer: "yes", "no", or "unknown" if it was not found before the Resources were exhausted. It also produces an Explanation of the Answer. Explanation is a list of notes supplied by the methods when activated during the problem-solving process. The main executional cycle is preformed by the procedure

applyknbase(Target, Resources, Target1, Resources1, Note)

It updates the target with new facts (producing Target1) and Resources, obtaining Resources1, i.e. resources left for the rest of the task.

```
problem:-
   initialize(Target, Resources),
   solve(Target,Resources,Answer,Explanation),
   display(Target,Answer,Explanation).

solve(Target,_,yes,nil) :-
   proved(Target).

solve(Target,_,no,nil) :-
   disproved(Target).

solve(Target,Resources,unknown,nil) :-
   exceeded(Resources,Target).

solve(Target,Resources,Answer,[Note | Exp1]) :-
   applyknbase(Target,Resources,Target1,Resources1,Note),
   solve(Target1,Resources1,Answer,Exp1).

applyknbase(T,Res,T1,Res1,Note) :-
   selectmethod(T,Mname,Context),
   execute(Mname,Context,Facts,Note,Spent),
   update(T,Facts,T1),
   subtract(Res,Spent,Res1).
```

Fig. 4 – PROLOG code for the top level operation of AL3.

4. REPRESENTATION OF TARGET AND FACTS

Target and facts are propositional calculus formulas. A target, T, can be thought of as a formula that the system is trying to prove or disprove. If T is a theorem then T has been proved; if not(T) is a theorem then T has been disproved; if T is neither of these then new facts, F, when found, are used as new axioms. The target T is updated by F giving a new target, $T1$:

$$T1 \equiv (F \Rightarrow T) \ .$$

Now the goal becomes to prove or disprove $T1$.

In the system, target, and facts are, to enable efficient manipulation, represented as sets of clauses (also called 'lines') of the form

$$a_1, a_2, \ldots, a_m \Rightarrow b_1, b_2, \ldots, b_n$$

meaning

$$a_1 \wedge a_2 \wedge \ldots \wedge a_m \Rightarrow b_1 \vee b_2 \vee \ldots \vee b_n \ .$$

All a_i and b_j are hypothesis names. The logical connective between clauses is conjunction. Any propositional formula can be converted into this form by Wang's algorithm (e.g. Raphael 1976). This form will be referred to as the 'c-form'.

Sometimes we will use the set notation in the following way. Capital letters A, B, ... will denote sets of hypothesis names. If $A = \{a_1, \ldots, a_m\}$ and $B = \{b_1, \ldots, b_n\}$ then

$$A \Rightarrow B$$

represents the line

$$a_1, \ldots, a_m \Rightarrow b_1, \ldots, b_n \ .$$

In this notation, a target in the c-form will be written as

$$\begin{bmatrix} A_1 \Rightarrow B_1 \\ A_2 \Rightarrow B_2 \\ \ldots\ldots\ldots \\ \ldots\ldots\ldots \end{bmatrix}$$

If $A_i = \phi$ then it represents the truth value "true". If $B_i = \phi$ then it represents the truth value "false".

A line $A \Rightarrow B$ is a *tautology* if

(1) A is false, or
(2) B is true, or
(3) $A \cap B \neq \phi$.

A target is *proved* if all its lines are tautologies (i.e. the target is a theorem).

A target can be decomposed into a product of its 'subtargets', where the subtargets themselves have the form of a target, e.g.:

$$T = T1 \times T2$$

The multiplication rule is:

$$T1 = \begin{bmatrix} A_1 \Rightarrow B_1 \\ \ldots\ldots\ldots \\ A_m \Rightarrow B_m \end{bmatrix}$$

$$T2 = \begin{bmatrix} C_1 \Rightarrow D_1 \\ \ldots\ldots\ldots \\ C_n \Rightarrow D_n \end{bmatrix}$$

$$T = T1 \times T2 = \begin{bmatrix} A_1 \cup C_1 \Rightarrow B_1 \cup D_1 \\ A_1 \cup C_2 \Rightarrow B_1 \cup D_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ A_1 \cup C_n \Rightarrow B_1 \cup D_n \\ A_2 \cup C_1 \Rightarrow B_2 \cup D_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ A_m \cup C_n \Rightarrow B_m \cup C_n \end{bmatrix}$$

A target

$$T = T_1 \times T_2 \times \ldots \times T_N$$

is a theorem if at least one of its subtargets T_i is a theorem. The multiplication operation is associative and commutative. These properties provide a basis for different strategies of problem decomposition.

An easy way of updating the target by new facts is through the use of multiplication. The principle is: to update a target T with facts F, we have T and F represented in the c-form by

$$[\text{true} \Rightarrow T]$$

and

$$[F \Rightarrow \text{false}] \ .$$

The updated target is then

$$[F \Rightarrow \text{false}] \times [\text{true} \Rightarrow T] = [F \Rightarrow T] \ .$$

It may be advantageous to keep the target in the product form delaying the multiplication, or to carry out the multiplication only on part of the target. For example, complete multiplication on a target

$$F1 \times F2 \times T$$

may result in a bulky and difficult to manipulate new target with many lines. Instead, a partial multiplication of $F1 \times F2 = F$ may reduce the number of lines, giving a handy new target represented by the product $F \times T$.

Another reason for keeping the target in the product form is that if the subtargets consist of basically disjoint sets of hypotheses then the product of the subtargets corresponds to a natural decomposition of the problem. Each subtarget then corresponds to a comparatively independent subproblem. This enables the system to focus its attention on subproblems themselves.

Facts of the form "Hypothesis a is true" or "a is false" can be added by simply substituting the value of a into the lines of the target and applying

simplification rules for logical expressions. Facts of more complex forms are transformed into the c-form and then added as a new multiplication factor. Thus for example a fact

$$a \Rightarrow b$$

is properly transformed into the c-form by the following operations. Factor to be added is

$$(a \Rightarrow b) \Rightarrow \text{false} \ .$$

Its c-form is obtained by the following transformations (using Wang's algorithm):

$$(\text{not}(a) \vee b) \Rightarrow \text{false} \ .$$

This is equivalent to

$$\text{not}(a) \Rightarrow \text{false} \quad \text{and}$$
$$b \Rightarrow \text{false}$$

giving finally

$$\begin{bmatrix} \text{true} \Rightarrow a \\ b \Rightarrow \text{false} \end{bmatrix}$$

Table 1 presents some useful transformations of typical forms of facts into a corresponding c-form representation.

The goal of the problem-solving process is to either prove the target or disprove it, that is, to demonstrate that the target is a theorem or that its negation is a theorem. Both alterations can be dealt with by keeping, during the problem-solving process, two targets: *positive* and *negative* target. If the positive target becomes a theorem then the initial target has been proved; if the negative target becomes a theorem then the initial target has been disproved. For example, assume that the initial goal was to answer the question: Is hypothesis h true or false? Then the corresponding positive and negative targets in the c-form are $[\text{true} \Rightarrow h]$ and $[h \Rightarrow \text{false}]$ respectively. New facts are, when generated, added multiplicatively to both positive and negative targets.

After inserting a truth value for a hypothesis name or after carrying out a multiplication operation, targets may become messy and redundant. They can be tidied up by applying the following simplification rules:

(1) Delete tautological lines.
A line $A \Rightarrow B$ is a tautology if $A \cap B \neq \phi$.
A line $\text{false} \Rightarrow B$ is a tautology.
A line $A \Rightarrow \text{true}$ is a tautology.

(2) Delete lines that are implied by other lines.
A line $A \Rightarrow B$ *implies* another line $A1 \Rightarrow B1$ within the same subtarget if $A \subseteq A1$ and $B \subseteq B1$.

Table 1 – Some useful transformations of facts into the c-form. For a given fact F, the right-hand side constructs in the table are logically equivalent to $F \Rightarrow$ false.

Fact	Fact in the c-form
$a \vee b \vee c \vee \ldots$	$a \Rightarrow$ false $b \Rightarrow$ false $c \Rightarrow$ false .....
$a \wedge b \wedge c \wedge \ldots$	$[a, b, c, \ldots \Rightarrow$ false$]$
$a \vee b \vee c \vee \ldots \Rightarrow h$	$h \Rightarrow$ false true $\Rightarrow a, b, c, \ldots$
$a \wedge b \wedge c \wedge \ldots \Rightarrow h$	$h \Rightarrow$ false true $\Rightarrow a$ true $\Rightarrow b$ true $\Rightarrow c$ .....
$a \vee b \vee c \vee \ldots \Leftarrow h$	true $\Rightarrow h, a, b, c, \ldots$ $a, h \Rightarrow$ false $b, h \Rightarrow$ false $c, h \Rightarrow$ false .....
$a \wedge b \wedge c \wedge \ldots \Leftarrow h$	$h, a, b, c, \ldots \Rightarrow$ false true $\Rightarrow a, h$ true $\Rightarrow b, h$ true $\Rightarrow c, h$ .....

(3) Insert truth values for hypothesis names whose truth value is implied by the targets.

The truth value of a hypothesis h is implied if h appears in the same side of all the lines of a positive (sub)target and in this same side of all the lines of a negative (sub)target. The value is:

(a) if h appears on the left then h is true;

(b) if h appears on the right then h is false.

This decision is based on the fact that positive and negative targets cannot both be theorems.

5. EXAMPLE: SEARCHING AND/OR GRAPHS IN AL3

This section presents a detailed example to illustrate the whole basic AL3 machinery at work. In the example we use a miniature knowledge-base for searching AND/OR graphs. The knowledge-base does not contain any heuristics to guide the search. It consists of one lemma, GOALTEST, and one method, EXPAND, whose detailed PROLOG definition is in Fig. 5. The preconditions for both is the existence of a hypothesis H:

goal(X) ?

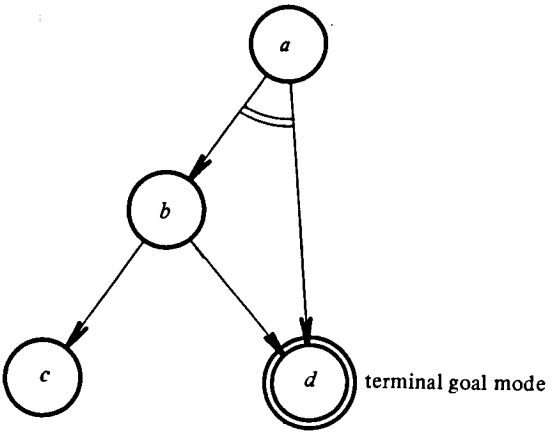
where X is a node in the AND/OR graph being searched, and goal(X) means that there exists a solution subgraph for X. If X is a terminal node which 'trivially' satisfies the goal-condition then the lemma returns the fact H is true.

```
    /*** Lemma GOALTEST ***/

precond(goaltest, [H,X]) :-
   hyp(H,goal(X)).

exec(goaltest,[H,X],[ [H] => false]) :- goalnode(X),!.

exec(goaltest,_,[true => false]).

    /*** Method EXPAND ***/

precond(expand,[H,X]) :-
   hyp(H,goal(X)).

exec(expand,[H,X],Fact) :-
   findall(Y,succ(X,Y),Ylist),
   (Ylist=[],!,Fact = [true => [H]];
    findall(Hname,newhyp(Hname,Ylist),Hlist).
    getfact(H,Hlist,X,Fact)).

newhyp(Hname,Ylist) :-
   member(Y,Ylist),
   genhyp(Hname,goal(Y)).

getfactH,Hlist,X,[([H|Hlist] => false)|Lines]) :-
   andnode(X),!,
   findall(true => [H,H1], member(H1,Hlist), Lines).

getfact(H,Hlist,X,[(true => [H|Hlist])|Lines]) :-
   ornode(X),!,
   findall([H,H1] => false, member(H1,Hlist), Lines).
```

Fig. 5 – PROLOG code of a knowledge-base for searching AND/OR graphs. The base assumes that the control module prefers lemmas to methods.

Let $Y_1, \ldots, Y_n$ be the successor nodes of a node X. The method EXPAND, when executed on the context [H,X], generates hypotheses $H_1, H_2, \ldots, H_n$ of the form

goal(Y_i) ?



Current Knowledge Pool	Next execution cycle
==========	==========
Positive Target	Conflict set (method names : contexts)
true=> [h:0]	1 expand: [h:0,a]
Negative Target	Which? 1.
[h:0] =>false	Executed expand on context [h:0,a]
Hypotheses	New facts
1 hyp (h:0,goal (a))	[h:0,h:1,h:2] =>false true=> [h:0,h:1] true=> [h:0,h:2]
Current Knowledge Pool	Next execution cycle
==========	==========
Positive Target	Conflict set (method names : contexts)
true=> [h:0,h:1] true=> [h:0,h:2]	1 goaltest: [h:1,b] 2 goaltest: [h:2,d] 3 expand: [h:1,b]
Negative Target	4 expand: [h:2,d]
[h:0,h:1,h:2] =>false	Which? 1.
Hypotheses	Executed goaltest on context [h:1,b]
1 hyp (h:0,goal (a)) 2 hyp (h:1,goal (b)) 3 hyp (h:2,goal (d))	New facts true=>false

Fig. 6 – Part of trace, produced by AL3, when searching the AND/OR graph in top of the figure.

In addition, new facts are generated, namely: If X has AND successors then the facts are:

$$H \Leftrightarrow H_1 \wedge \ldots \wedge H_n$$

If X has OR successors then the facts are

$$H \Leftrightarrow H_1 \vee \ldots \vee H_n$$

The PROLOG code in Fig. 5 generates these facts already transformed into a proper form according to the transformation rules in Table 1. Fig. 6 is part of the trace, produced by AL3, when solving the problem

goal(a) ?

for the AND/OR graph in top part of Fig. 6.

6. CONCEPTS MANIPULATED BY AL3

In principle, the AL3 system as described in the previous sections is not limited to any special formalism for representing methods for problem-solving, or to any special class of concepts to be used for solving problems. In this section we present a formalism and a number of concepts that are useful for solving combinatorial problems in general and chess problems in particular. These concepts were used in the experiments with AL3 on chess endgames.

6.1 Piece-of-advice

A fundamental concept of AL1, piece-of-advice, proved to be extremely valuable, not only for representing knowledge, but also because it provides a good formal basis for precise definition of other concepts. A *piece-of-advice*, A, is a five-tuple

$$(X, BG, HG, MCX, MCY)$$

where X is the side (White or Black) to which A belongs, BG and HG are predicates on positions called *better-goal* and *holding-goal* respectively, MCX and MCY are predicates on moves, called *move-constraints* for side X and side Y respectively. Throughout the paper we use X and Y to represent both sides. Thus X can be either White or Black, and Y is always the opponent of X. Besides a mere selection of a subset of legal moves, move-constraints can impose an ordering on the moves that are selected. This becomes important for practical reasons when searching a game-tree.

A tree T is called a forcing-tree for a piece-of-advice

$$A = (X, BG, HG, MCX, MCY)$$

in a position *Pos*, iff T is a subtree of the game-tree rooted in *Pos*, such that:

(1) for every node p in T: $HG(p)$;
(2) for every nonterminal node p in T: not $BG(p)$;
(3) for every terminal node p in T: $BG(p)$ or p is a Y-to-move position from which there is no legal move that satisfies MCY;

(4) there is exactly one move in T from every X-to-move nonterminal node in T; that move must satisfy MCX;
(5) all legal moves from any nonterminal Y-to-move position in T that satisfy MCY are in T.

A piece-of-advice A is *satisfiable* in a position Pos iff there exists a forcing-tree for A in Pos. We write:

$$\text{sat}(A, Pos)$$

In fact, a piece-of-advice defines a subgame with two possible outcomes: win or loss. Legal moves of this subgame are defined by the move-constraints, and terminal positions of the subgame are defined by predicates better-goal and holding-goal. A position Pos is won for side X with respect to the subgame corresponding to a piece-of-advice A if $\text{sat}(A,Pos)$; otherwise it is lost for X with respect to A.

Note an important detail in the above definition of forcing-tree: if (1) $HG(Pos)$ and $\text{not}(BG(Pos))$, and (2) Y-to-move in Pos, and (3) no Y-move in Pos satisfies MCY then Pos is terminal node of a forcing tree. This interpretation of the 'no-move' condition ensures the following relation for any piece-of-advice and any position:

$$\text{sat}((X,BGX,HGX,MCX,MCY),Pos) \Leftrightarrow$$
$$\text{not sat}((Y,\text{not}(HGX),\text{not}(BGX \text{ and } HGX),MCY,MCX),Pos)$$

This relationship will be referred to as 'inverse-advice relationship'. However, this definition sometimes necessitates that the test for stalemate in chess is explicitly stated in the goals of a piece-of-advice to avoid anomalous behaviour.

Some other useful relations concering the satisfiability of related pieces-of-advice are given in Bratko & Niblett (1979).

6.2 Plans

A *plan*, P, is a quadruple

$$(X, BG, HG, MCX)$$

where X is the side to which P belongs, BG and HG are predicates on positions called better-goal and holding-goal, and MCX (move-constraints for side X) is any schema for selecting and/or ordering X moves. An example of such move-constraints is a White king 'macromove' between two specified squares not exceeding a specified length, e.g.:

$$\text{macromove(white-king, c2, e1, length} \leqslant 3) \ .$$

This macromove denotes the set of all possible king-paths between squares c2 and e1 of the length of at most 3 moves (see Fig. 7). A plan with such move-constraints allows any legal White king move along this macromove.

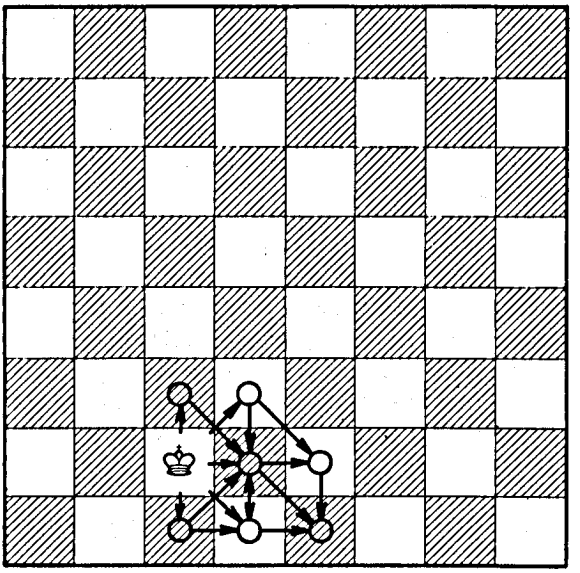


Fig. 7 – White king macromove from c2 to e1 in at most 3 moves.

Move-constraints can also prescribe an ordering of moves. For example the above king macromove can be ordered with respect to increasing length of the king-paths.

We say that a plan $P = (X, BG, HG, MCX)$ *succeeds* in a position *Pos* iff

$$\text{sat}(A, Pos)$$

where A is a piece-of-advice

$$A = (A, BG, HG, MCX, \text{anymove}) \ .$$

We write

$$\text{suc}(P, Pos) \Leftrightarrow \text{sat}(A, Pos) \ .$$

Let *Px* and *Py* be two plans

$$Px = (X, BGX, HGX, MCX)$$
$$Py = (Y, BGY, HGY, MCY) \ .$$

Then plan *Py refutes* plan *Px* in a position *Pos* iff

$$\text{not sat}(A, Pos)$$

where A is a piece-of-advice

$$A = (X, BGX \text{ or not}(HGY), HGX, MCX, MCY) \ .$$

We write

$$\text{ref}(Py, Px, Pos) \Leftrightarrow \text{not sat}(A, Pos) \ .$$

An equivalent definition is

$$\text{ref}(Py, Px, Pos) \Leftrightarrow \text{sat}(A1, Pos)$$

where $A1 = (Y, \text{not}(HGX), \text{not}(HGX) \text{ or } \text{not}(BGX) \text{ and } HGY, MCY, MCX)$. The equivalence of these two definitions can be proved by using the inverse-advice relationship.

Note that it is possible that a plan Py does not succeed (i.e. its better-goal is not attainable), but Py may still refute a plan Px.

6.3 "Or" combination of plans

Let $P1$ and $P2$ be plans

$$P1 = (X, BG1, HG1, MCX1)$$
$$P2 = (X, BG2, HG2, MCX2) \ .$$

A plan

$$P = P1 \text{ or } P2$$

is called an 'or-combination' of $P1$ and $P2$. The better-goal of P is $BG1$ or $BG2$. Precise combinations of the goals and move-constraints of $P1$ and $P2$ can be defined by the following AL3 method for investigating the success of or-plans.

Method OREXPAND

Precondition

$$\text{hypothesis}(H, \text{suc}(P1 \text{ or } P2, Pos)),$$
$$P1 = (X, BG1, HG1, MCX1)$$
$$P2 = (X, BG2, HG2, MCX2)$$

Action

1. Generate hypotheses: $\text{hypothesis}(Ha, \text{suc}(P1, Pos))$,
   $\text{hypothesis}(Hb, \text{suc}(P2, Pos))$
2. If Y-to-move in Pos then generate all legal successor positions $Pos_1, \ldots, Pos_n$ of Pos, else generate legal successor positions $Pos_1, \ldots, Pos_n$ such that the moves $Pos \rightarrow Pos_i$ satisfy either $MCX1$ or $MCX2$ or both. The ordering of $Pos_1, \ldots, Pos_n$ is: rough ordering by the criterion "first satisfy both move-contraints", and fine ordering as prescribed by $MCX1$ and $MCX2$.
3. Generate:
   $\text{hypothesis}(H1, \text{suc}(P1 \text{ or } P2, Pos_1))$
   .....
   $\text{hypothesis}(Hn, \text{suc}(P1 \text{ or } P2, Pos_n))$
4. Return facts
   (a) if X-to-move in Pos then
   $$Ha \vee Hb \Rightarrow H1 \vee H2 \vee \ldots \vee Hn$$
   $$Ha \vee Hb \vee H1 \vee \ldots \vee Hn \Leftrightarrow H$$
   (b) if Y-to-move in Pos then
   $$Ha \vee Hb \Rightarrow H1 \wedge H2 \wedge \ldots \wedge Hn$$
   $$Ha \vee Hb \vee (H1 \wedge \ldots \wedge Hn) \Leftrightarrow H \ .$$

6.4 Modification of plan by plan

Let $P1$ and $P2$ be two plans for side X. The modification of $P1$ by $P2$ is a plan $P = P1 \bmod P2$, such that the goal of P is the same as the goal of $P1$, but the sequence of steps of $P1$ may be interrupted by inserting steps of $P2$. An AL3 method for investigating the success of a modified plan is:

Method MODEXPAND
Precondition

hypothesis(H, suc($P1 \bmod P2, Pos$)),
$P1 = (X, BG1, HG1, MCX1)$,
$P2 = (X, BG2, HG2, MCX2)$

Action

1. Generate: Hypothesis(Ha, suc($P1, Pos$)).
2. If Y-to-move in Pos then generate all legal successors $Pos_1, \ldots, Pos_n$ of Pos else generate legal successors $Pos_1, \ldots, Pos_n$ satisfying $MCX1$ or $MCX2$ or both. The ordering of $Pos_1, \ldots, Pos_n$ is: rough ordering by "first satisfy both move-constraints", then "satisfy $MCX1$" then "satisfy $MCX2$"; fine ordering as prescribed by $MCX1$ and $MCX2$.
3. Generate:
   hypothesis($H1$, suc($P1 \bmod P2, Pos_1$))
   .....
   hypothesis(Hn, suc($P1 \bmod P2, Pos_n$))
4. Return facts:
   (a) if X-to-move in Pos then
   $Ha \Rightarrow H1 \vee \ldots \vee Hn$
   $Ha \vee H1 \vee \ldots \vee Hn \Leftrightarrow H$
   (b) if Y-to-move in Pos then
   $Ha \Rightarrow H1 \wedge \ldots \wedge Hn$
   $Ha \vee (H1 \wedge \ldots \wedge Hn) \Leftrightarrow H$.

7. A KPKP KNOWLEDGE-BASE

Here we outline a small AL3 knowledge-base for the king and pawn *vs.* king and pawn chess ending with both pawns passed (pawns not on the same file or on adjacent files). Correct play in this ending can be very difficult, as indicated by many chess studies from this domain (e.g. Averbach & Maizelis 1974). An example is in Fig. 1.

The KPKP knowledge-base contains two lemmas. One, CATCHPAWN, decides whether a king can stop a running opponent's pawn. The other, PAWNRACE, decides which pawn wins a pawn-race.

The methods in the base implement basic motifs of the KPKP ending with passed pawns, and some more general, "meta-level" ideas about plans. The following is an informal description of the most important methods. Appropriate

facts generated by the methods are obvious. At present the KPKP knowledge-base is not complete in the sense of producing correct play in all positions from this domain.

Method WIN

To win it is necessary to queen the pawn and not allow the opponent's pawn to queen successfully.

Method DRAW

To investigate the question "Can one side draw?", consider the question" Can the other side win?".

Method PUSH

One plan for queening is to push the pawn.

Method STOPPAWN

One way of preventing an opponent's plan to queen the pawn is to stop it by the king. It is assumed that the capture of that pawn implies that the pawn has been stopped.

Method RACE

One counter-plan against a queening plan is to queen own pawn.

Method STOPKING

If an opponent's plan consists of a king-macromove, then it may be refuted by a king's intervention: own king-macromove intersecting the opponent's king-macromove.

Method MODIFYPLAN

If a plan $P1$ fails against an opponent's plan R then $P1$ may be successfully improved in the following way: find a plan, $P2$, which, hopefully, refutes the plan R, and propose a new plan: $P1$ modified by $P2$, i.e. $P = P1\ mod\ P2$. To find $P2$, AL3 solves a local subproblem with its own local target of refuting R. To solve the local subproblem AL3 may use all the knowledge in the knowledge-base.

Method ORPLAN

If two plans, $R1$ and $R2$, are known not to refute an opponent's plan P, then the or-combination of $R1$ and $R2$, i.e. $R = R1\ or\ R2$, may refute P.

Method SEARCH

This method converts a hypothesis of the form suc(*Plan*, *Pos*) or ref(*Plan*1, *Plan*2, *Pos*) into sat(A, *Pos*) where A is a corresponding piece-of-advice, and checks the satisfiability of A by searching the game-tree. This method can be very expensive and is therefore used only occasionally.

Method EXPAND

Expands a hypothesis of the form suc(*Plan*, *Pos*) by generating successor positions

of *Pos*, subject to move-contraints in *Plan*, and hypotheses that *Plan* succeeds in the successor positions. New facts are generated according to who is to move in *Pos* and according to the form of *Plan* (see expansion rules for *P*1 *mod P*2 and *P*1 *or P*2 in the previous section).

To illustrate how this knowledge-base works, consider the position in Fig. 8. AL3 is asked the question "Can White win in this position?". Fig. 9 shows the current-knowledge-pool at the moment when AL3 has found a correct plan for White to win. This is: queen the White pawn by pushing it and in the meantime stop the Black pawn with the White king if necessary.

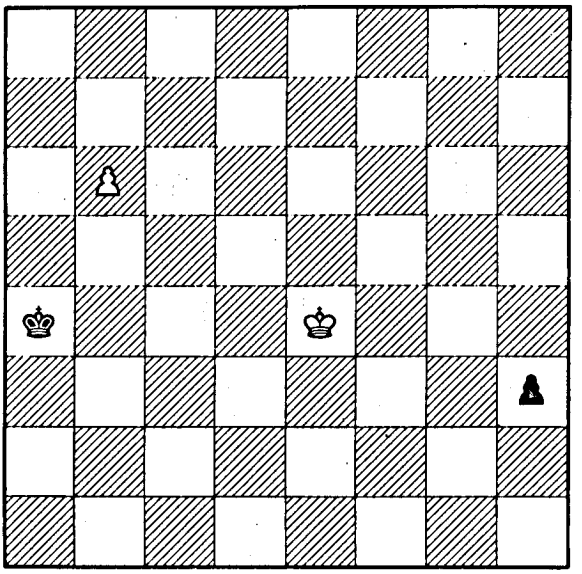


Fig. 8 – White to move, can White win? Correct is Ke4–f3 stopping the Black pawn. After that, the White pawn cannot be stopped.

8. DISCUSSION

There are several ways of looking at the AL3 system. One possible view is that AL3 is a theorem-prover which accepts a problem in the form of a formula to be proved or disproved. If the formula is neither a theorem nor a contradiction then AL3 tries to find new facts about the problem. The new facts are then used as additional axioms for proving or disproving the initial formula. In this sense each executional cycle aims at producing the most useful new axioms, such that they bring the formula as close as possible towards a theorem or a contradiction.

Another view of AL3 is that AL3 is a problem-solver which uses a special formalism for problem representation. This formalism can be thought of as a generalization of two known schemas for problem-representation: the state-space representation and the AND/OR graph representation (e.g. Nilsson 1971). With

```
Current Knowledge Pool
======================

Postive Target

[h:4,h:5] => [h:0,h:1,h:2,h:3,h:8]

Negative Target

[h:0,h:1,h:4,h:5] => [h:2,h:3]

Facts

1 h:3 is false
2 h:5 is true
3 h:7 is true

Hypotheses

1 hyp(h:0,canwin(w,pos))
2 hyp(h:1,suc(p:1,pos))
3 hyp(h:2,suc(p:2,pos))
4 hyp(h:3,ref(p:3,p:2,pos))
5 hyp(h:4,ref(p:4,p:2,pos))
6 hyp(h:5,ref(p:5,p:2,pos))
7 hyp(h:6,suc(p:5,pos))
8 hyp(h:7,ref(p:6,p:5,pos))
9 hyp(h:8,suc(p:2 mod p:6,pos))

Plans

1 plan(p:1,w,queenwin(w),alive(w) & (not queendraw(b)),any)
2 plan(p:2,w,queenwin(w),alive(w) & (not queendraw(b)),push(w))
3 plan(p:3,b,stopped(w),nil,macro(b,k,path(w,p)))
4 plan(p:4,b,queendraw(b),alive(b),any)
5 plan(p:5,b,queendraw(b),alive(b),push(b))
6 plan(p:6,w,stopped(b),nil,macro(w,k,path(b,p)))
```

Fig. 9 – AL3's current-knowledge-pool at the moment when a correct plan has been found for the position in Fig. 8. The symbol "pos" denotes that position. Goals in the plans mean: queenwin(w): White pawn has queened and position is won for White; queendraw(b): Black pawn queened and position not lost for Black; alive(w): White pawn not captured; etc.

respect to the logical relationships among the nodes in the problem space, the state-space representation could be called an 'OR-graph' representation, because all the sibling nodes in the state-space are disjunctively related. In this sense, the AND/OR-graph representation is a generalization of the state-space representation. Further, AL3's representation is a generalization of AND/OR-graph representation, and could be therefore called a 'general graph' representation, 'general' because it allows any logical relationship between the neighbouring nodes in the problem-space. This logical relationship is defined by new facts that can be any propositional calculus formula.

AL3, viewed as above, solves problems by searching a problem-space that does not consist of objects, defined by the problem-domain itself (e.g. rules of the game), but also of higher-order concepts like plans, macromoves, and pieces-of-advice. There is no formal distinction in the knowledge-base of AL3 between rules of the game and knowledge about how to solve problems.

Very little has been said about the control module of AL3 which implements the overall problem-solving strategy. One such strategic decision is whether to keep the current target in the form of a product or to carry out the multiplication, or to do the multiplication only partially, on some of the subtargets. In the experiments with AL3, described in this paper, the control module used the following simple strategy:

1. Carry out every multiplication in the target immediately.
2. Find 'interesting' hypotheses by simply counting the number of hypothesis occurrences in the left- and right-hand sides of the lines in the target. Hypotheses with high frequencies are interesting. Include in the conflict set only lemmas and methods that are applicable to interesting hypotheses, and that produce 'complementary' facts (that is: if an interesting hypothesis tends to occur on the left-hand side in the target then a complementary fact contains this hypothesis on the right-hand side).
3. Choose a lemma or a method from the conflict set in the following order of preference: first lemmas, then easy methods, then difficult methods (methods in the knowledge-base are characterised by 'easy' or 'difficult').

Design of more sophisticated control strategies seems to be necessary for solving larger-scale problems. One way of improving the above simple strategy is to delay (partially) the multiplication operation when updating the target and thus control the growth of the target. Another improvement, aiming at the reduction of the possibly very time-consuming matching of method-preconditions against the complete CKP is to limit this matching to a 'window' in CKP only. The window consists of the hypotheses dealt with in the previous executional cycle and their neighbouring hypothesis. Thus the window provides a mechanism for focusing AL3's attention to a part of CKP.

Another interesting problem for further experiments is concerned with the inclusion of more 'meta-knowledge' into the knowledge-base to facilitate more sophisticated reasoning about plans and pieces-of-advice. Such knowledge could provide rules for deciding whether a given plan, $P1$ say, is more specific than another plan, $P2$; then if yes and if $P2$ is known to fail then $P1$ also fails. Even very simple 'meta-methods' in the KPKP knowledge-base are sufficient for discovering concepts like a joint action of a king and a pawn. For example, if White's plan is to queen his pawn by pushing the pawn, and a Black king-macromove refutes this plan by stopping the White pawn, the White's plan can be modified by a White king-macromove preventing the Black king-macromove. This

effectively results in the idea: support the advancement of the White pawn by the White king.

Acknowledgements

The author would like to thank Professor Donald Michie for encouragement and support, T. B. Niblett for collaboration at the previous stages of this research, and the Edinburgh PROLOG development group (L. Byrd, F. Pereira and D. Warren) for their continuous advice on the PROLOG implementation of AL3. The following institutions made this work possible by providing financial and other support: the Science Research Council, UK; the Machine Intelligence Research Unit, University of Edinburgh, UK; the Faculty of Electrical Engineering and Jozef Stefan Institute, Ljubljana, Yugoslavia.

References

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

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

Bratko, I., & Michie, D., (1980b). An advice-program for a complex chess-programming task, *Computer Journal*, 23, 353-350.

Bratko, I., & Niblett, T., (1979). Conjectures and refutations in a framework for chess end-game knowledge, in *Expert Systems in the Microelectronic Age*, 83-102, (ed. Michie, D.). Edinburgh: Edinburgh University Press.

Michie, D., (1976). An advice-taking system for computer chess. *Comp. Bull.*, 11, 12-14.

Michie, D., (1980). Problems of the conceptual interface between machine and human problem-solvers, *Experimental Programming Reports No. 36.* Edinburgh: Machine Intelligence Research Unit, University of Edinburgh. To appear in *Prospects for Man: Computers and Society.* Toronto: York University.

Mozetic, I., (1979). Advice Language and the AL 1.5 program system. Work report and manual. Ljubljana: Jozef Stefan Institute. A shortened version appears as *Research Memorandum MIP-R-130.* Edinburgh: Machine Intelligence Research Unit, University of Edinburgh.

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

Pereira, L. M., Pereira, F. C. N., & Warren, D. H. D., (1978). User's guide to DECsystem-10 PROLOG. Edinburgh: Department of Artificial Intelligence, University of Edinburgh.

Pitrat, J., (1977). A chess combinations program which uses plans. *Artificial Intelligence*, 8, 275-321.

Raphael, B., (1976). *The Thinking Computer: Mind Inside Matter.* San Francisco: Freeman.

Tan, S. T., (1977). Describing pawn structures, in *Advances in Computer Chess*, 1, 74-88, (ed. Clarke, M. R. B.). Edinburgh: Edinburgh University Press.

Waterman D. A., & Hayes-Roth, F., (eds.) (1978). *Pattern-Directed Inference Systems.* New York: Academic Press.

Wilkins, D., (1979). Using patterns to solve problems and control search, Ph.D.Thesis. Stanford: Computer Science Department, Stanford University.

APPENDIX: SOME DETAILS OF AL3's PERFORMANCE ON KPKP

The KPKP knowledge base, outlined in section 7, also contains methods which employ the game-tree search. Using these methods, AL3 can of course, in principle, solve any KPKP problem. However, a straightforward application of search would

be rather complex: in difficult KPKP positions, the depth of search required would be at least 15 ply with the branching factor of about 8 (before pawns become queens, and much more afterwards) in the space of at least several hundred thousands of positions. Owing to the inefficiency of the present PROLOG implementation of AL3, searches of this size are prohibitive. However, it is interesting to see what AL3 can do if the search is used only very modestly, in particular if the numbers of nodes searched are of a similar order of magnitude to that known for human chess masters. This constraint is interesting not only for the sake of efficiency, but also because it ensures that the program's behaviour fits the 'human window' (D. Michie 1980). The application of the search methods was, in the experiments reported here, constrained so that any search was limited to at most 100 (or sometimes 200) nodes, i.e. 100 chess positions.

Comparatively simple positions, like the one in Fig. 8, present no problems to AL3 under this constraint. The system easily finds the correct main idea and is also able to work out all the tactical details up to the decisive queening of pawns. In difficult KPKP positions, like the Reti study in Fig. 1, this search constraint can make the system behave less confidently. Fig. 10 shows the AL3's CKP after 15 main executional cycles when solving the Reti study. At this moment, AL3 knows that:

(1) The White king alone cannot catch the Black pawn.
(2) Black, however, cannot simply push his pawn because in that case the White pawn wins the pawn-race. Therefore the advancement of the Black pawn is necessarily slowed down because the Black king has in the meantime to stop the White pawn.
(3) The White pawn alone cannot save the draw; but it is not clear whether a joint action of White pawn and White king to promote the pawn refutes Black's plan.

So far AL3 has tried to investigate four hypotheses by search: $h4$, $h10$, $h12$, and $h13$. Three times the search was carried out successfully within the search limit. Complexities of these searches were: $h4$: 3 nodes, $h10$: 6 nodes, and $h13$: 35 nodes. The remaining search failed to produce a fact as the search budget was exhausted before a result was obtained. At this point the plans for both sides have become too complex to be investigated by search limited to 100 nodes. Therefore the system constructs the 'state-of-the-art' hypothesis by combining all currently known and not yet refuted ideas for both sides. The state-of-the-art hypothesis here becomes:

$$\text{ref}(p2 \text{ or } p3 \bmod p6 \text{ or } p6, p1 \bmod p4, \text{reti}) = ?$$

This in fact contains the best plans for both sides. The system now tries to investigate this question by searching to the greatest depth that is still doable within the 100 node search limit. It turns out that search to the depth of 9 ply is still doable under this constraint. The search takes 78 nodes and produces a forcing-tree consisting of 57 moves. This forcing-tree is proposed by the system

```
Current Knowledge Pool
======================

Positive Target

[h:1]=>[h:0,h:2,h:8]
[h:1]=>[h:0,h:2,h:12,h:14]

Negative Target

[h:0]=>[h:1,h:2,h:8]

Facts

1 h:3 is false
2 h:4 is true
3 h:10 is false
4 h:13 is false

Hypotheses

1 hyp(h:0,candraw(w,reti))
2 hyp(h:1,canwin(b,reti))
3 hyp(h:2,suc(p:1,reti))
4 hyp(h:3,ref(p:2,p:1,reti))
5 hyp(h:4,ref(p:3,p:1,reti))
6 hyp(h:5,suc(p:3,reti))
7 hyp(h:6,ref(p:4,p:3,reti))
8 hyp(h:7,ref(p:5,p:3,reti))
9 hyp(h:8,suc(p:1 mod p:4,reti))
10 hyp(h:9,suc(p:4,reti))
11 hyp(h:10,ref(p:3,p:1 mod p:4,reti))
12 hyp(h:11,ref(p:6,p:4,reti))
13 hyp(h:12,ref(p:3 mod p:6,p:1 mod p:4,reti))
14 hyp(h:13,ref(p:2,p:1 mod p:4,reti))
15 hyp(h:14,ref(p:6,p:1 mod p:4,reti))

Plans

1 plan(p:1,b,queenwin(b) or easywin(b),not pexposed(b) and
  not easystop(b) and not qeendraw(w) and not easywin(w),push(b))
2 plan(p:2,w,fail,nil,catchmacro(w,k))
3 plan(p:3,w,queendraw(w),alive(w),push(w))
4 plan(p:4,b,fail,nil,catchmacro(b,k))
5 plan(p:5,b,queenwin(b),alive(b),push(b))
6 plan(p:6,w,fail,nil,macro(w,k,8..8,stopset(macro(b,k,1..6,path(w,p),
  l=<2,shortestfirst)),l=<6,shortestfirst))
```

Fig. 10 - CKP after 15 executional cycles when solving the Reti study.

as the 'best try' for both sides. It contains the correct move for White, Kg7, which draws. The tree also foresees the two critical variations starting by two of Black's replies: h4 and Kb6. The forcing-tree indicates the intention to answer the move h4 by the correct Kf6, preserving the draw, but Kb6 is intended to be answered by Kg6 which loses. Thus although the system plays the correct move Kg7 this result is not perfect as the system does not predict the correct further

play in all variations. Indeed, in the variation 1 Kg7 Kb6 the system plays the losing 2 Kg6. However, if the constraint on search is relaxed by shifting the search limit to 200 nodes, the system plays correctly in all variations starting from the Reti position in Fig. 1. This new search limit allows the system to generate a forcing-tree with the correct reaction to Kb6 in the variation: 1 Kg7 Kb6, 2 Kf6 h4, 3 Ke5! h3, 4 Kd6 etc. The complexity of this critical search is 184 nodes.

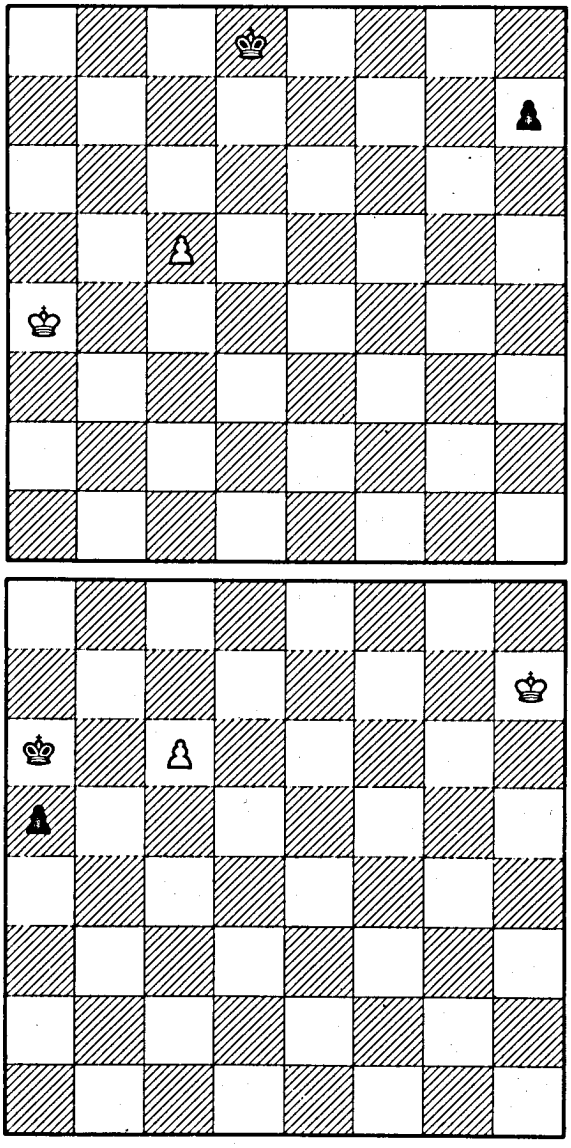


Fig. 11 – A study by Moravec (first position) and a study by Adamson. Both: White to move and draw.

Fig. 11 shows two other difficult examples in which AL3 behaves similarly as in the Reti study. It correctly finds the plans for both sides, and also finds the best first two moves for White within the search limit of 100 nodes. Again, the search-constraint makes it impossible to work out all the details up to obviously drawn positions. In the study by Moravec the system tries seven times to solve subproblems by game-tree search. Five times the search produces an answer (search statistics for these five searches are: 81 nodes, 6 nodes, 5 nodes, 59 nodes, and 86 nodes). Twice the search fails to produce a result before 100 nodes have been generated. This is, however, sufficient for the system to propose the following pretty manoeuvre of the White king which preserves the draw: 1 Kb5 h5, 2 Kc6! (with the idea 2 ... h4, 3 Kb7 followed by the advancement of the White pawn), or if 1 ... Kc7 then 2 Kc4 with a draw.

Similarly in the study by Adamson, the system proposes as the best lines for both sides a forcing-tree which contains the solution of the study: 1 Kg6 a4 (or Kb6), 2 Kf5. Again, owing to the 100 node limit, the forcing-tree does not predict all the lines up to obviously drawn positions.

In the examples so far, the system was, although impeded by the search-limit, able to find correct ideas and correct play. However, an example has been found (Fig. 12) which shows how the 100 node search-limit can lead to serious troubles. Difficult positions require the combination of several ideas by combining several plans in appropriate ways to find composed plans which actually work. As these plans become more complicated, they are harder to verify

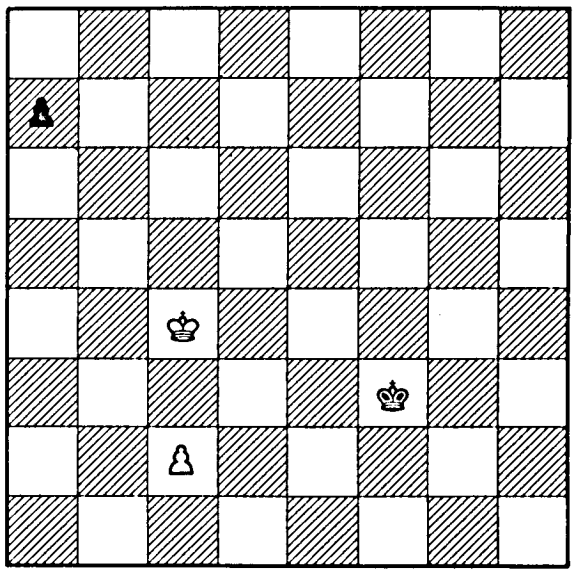


Fig. 12 – A study by Grigoriev: White to move and win. Solution is: 1 Kd4! Kf4, 2 c4 Kf5, 3 Kd5 etc. The idea of 1 Kd4! is to open the path of the White pawn, to prevent the Black king from getting in front of the White pawn, and to retain an option for the White king to stop the Black pawn at square a1.

by search. This can prevent the system from discovering some useful fact during the problem solving process. In a study by Grigoriev in Fig. 12, the system discovers that White has to advance his pawn and try to prevent the Black king from stopping this pawn, and that Black has to stop the White pawn by the king or advance his own pawn. But there are not enough facts known to motivate the system to consider another vital component of the correct White plan: White king has also to guard the Black pawn. The move which AL3 finally proposes is c3 which allows Black to draw.