BRIAN EDWIN PETER ALDEN, B.A.

AN EXPERT SYSTEM APPROACH TO RETROGRADE-ANALYSIS

Thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
in the
FACULTY OF MATHEMATICS of
THE OPEN UNIVERSITY
February 1985

ABSTRACT

This thesis presents a system called RETRO, which has the capability of solving a number of retrograde-analysis chess problems of varying degrees of difficulty. Retrograde-analysis problems are exercises in deductive reasoning, entirely different from conventional 'mate-in-n-moves' chess problems, their nature being such that any type of question, together with perhaps an assortment of associated initial conditions, may be presented to the solver, whilst their solution lies in asking pertinent questions at each step of the deduction process.

The aim has been to investigate the extent to which the deduction of past events may direct attention to these questions, a key concept being that of a 'Significant Event' (SE), which is an event of significance that must have occurred at some time in the history of the game. An SE is akin to a frame, the usual slots being replaced by the questions that the SE prompts, to which the answers may either provide a solution to the original problem or point to another SE, together with its association questions. It is shown that a small number of SEs possess widespread application.

RETRO maintains an explicit knowledge base of rules, grouped according to function, which are used both by the SEs and the system. This knowledge base is designed for easy amendment.

RETRO also maintains an explanation facility, which provides an account of its deductions, including a listing of rules utilised.

A number of examples are presented which demonstrate the effectiveness of the SE approach. There is also a discussion as to how the concept may be expanded, and its possible limitations.

ACKNOWLEDGEMENTS

Stanley Collings, for inspiring the interest in retrograde-analysis.

Dr. Max Bramer, my supervisor, for his unreserved encouragement and support.

The Academic Computing Service of the Open University.

Vic Clouston and the staff of the Open University Regional Office at Bristol for all their kindness and help.

Raymond Smullyan, without whose writings this project would have been much more difficult.

Father Aloysius Hacker, without whose spiritual guidance and advice I would not have succeeded.

TABLE OF CONTENTS

1.0 INTRODUCTION

1.1 The work described in this thesis is concerned with the solution of a specialised form of problem (closely concerned with the game of chess) by means of heuristic methods. The thesis defines an expert system called RETRO, whose domain of application is retrograde-analysis chess problems. This type of problem, chess logic problems, as they are sometimes called, differs from the conventional type of chess problem in that it is concerned only with the past history of the game, and what may be deduced about it. To understand better the tenor of the work some knowledge of retrograde-analysis is required, an appetising source of such knowledge being Raymond Smullyan's delightful book, The Chess Mysteries of Sherlock Holmes. The author is particularly indebted to this book, both as a source of inspiration and a source of problems.

Those who may be meeting retrograde-analysis for the first time will find a gentle introduction to the subject in Section 1.4, where Mr. Sherlock Holmes himself has consented to be the guide. Mr. Holmes will in fact be making an occasional appearance in these pages, to elucidate, and to challenge the Reader.

1.2 Information-processing

It is a major paradigm of Artificial Intelligence (A.I.) that intelligent processes may be mechanised and that by using programs as tools in their study we may perhaps become better able to understand their nature. Winston [22] defines the central goals of A.I. as making computers more useful and understanding the principles which make intelligence possible, whilst Marr [18] defines A.I. as the study of complex information-processing problems that have their roots in some aspect of biological information-processing. Marr further defines the goal of A.I. as identifying a useful information-

processing problem and accounting for how it was solved.

An important development, arising largely from A.I. research, is the idea of an expert system, a useful survey and critical review of which may be found in Bramer [19], who offers the following definition:

> 'An expert system may be defined as a computing system which embodies organised knowledge concerning some specific area of human expertise, sufficient to perform as a skilful and cost-effective consultant.'

There is no doubt that much expert knowledge will be heuristic and ill-defined by nature, with the expert being a master in the art of good guesswork. The problem of how knowledge must be represented in order to achieve expert performance has been the focus of much research, ranging from systems using production rules (e.g. Shortliffe [5]) and frames (e.g. Goldstein and Roberts [20]) to combinations of knowledge representations.

The solution of retrograde-analysis problems may be regarded as an information-processing problem, and, with RETRO, steps towards its solution are presented.

> 'There are occasional chess situations, Watson, which challenge the analytic mind as fully as any which arise in real life. Moreover, I have found them as valuable as any exercises I know in developing those powers of pure deduction so essential to dealing with real-life situations.' S. Holmes (from [1], p. 3).

It is not claimed that the solution is complete, or indeed that it is the only solution. What is presented here is a report of an actual implementation, not an idealised 'Version N'. RETRO utilises the frame concept to direct attention to the questions that must be asked in order to effect a solution, and reasons are given to show that the approach taken is of general validity in its domain of interest. Also discussed is the question: 'What happens if RETRO is unable to find a solution?' (It does <u>not</u> enter an infinite loop.)

1.3 Layout of thesis

To those who may be new to the concept of retrograde-analysis the following section (1.4) is devoted. It offers a gentle introduction to the subject, with the benefit of advice from Mr. Sherlock Holmes.

This is followed by an overview of RETRO (Chapter 2), in which the essential features of the program are reviewed and the basic concept of Significant Events (SEs) introduced. A block diagram is presented, together with a listing of the more important functions and variables. An example of RETRO at work is given, followed by an analogy with the basic SE concept. Finally in this chapter a number of retrograde-analysis problems and their RETRO answers are given; the reader may care to consider these before meeting the complete solutions in later pages.

Chapter 3 takes a deeper look at retrograde-analysis, considering the type of questions (and their initial conditions) that may be asked. Ways of classifying these problems are discussed, followed by an introduction to Significant Events.

SEs are examined further in Chapter 4, together with examples of the rule structure used by RETRO.

Chapter 5 deals with control, plus a discussion of what happens if RETRO is unable to find a solution, and what happens if RETRO meets an illegal situation (always a possibility in retrograde-analysis).

Knowledge acquisition and explanation are covered in Chapter 6, which includes a discussion of the rule groupings used in RETRO, and the use of a dynamic data-base to facilitate explanation.

Chapter 7 consists of a number of RETRO examples.

An overview and critical review of related work are given in Chapter 8, comparisons with RETRO being made.

Some other expert systems are reviewed in Chapter 9, with particular reference to RETRO. A look is then taken at the limitations of RETRO.

Chapter 10 reviews a host of recently published retrograde-analysis problems by Raymond Smullyan [2], referring in particular to the SE concept as applied to these problems.

Final conclusions are drawn in Chapter 11 with respect to knowledge representation, control, knowledge acquisition and explanation. Possible future work is also discussed.

There are a number of Appendices, which include listings of the rules used by RETRO and some of the more important functions.

1.4 Retrograde-analysis - an introduction

Retrograde-analysis (or retro-analysis) problems are problems where deductions must be made about the past history of the game.

These problems conform to the following pattern:

a. A board position is given.

b. Initial data may be provided: for example, 'Black moved last'.

c. A question is asked: for example, 'Can Black castle?'.

In order to answer the question trails must be sought, trails that point to a sequence of events and/or moves that must have occurred at some time during the course of the game. Some ways of uncovering these trails are:

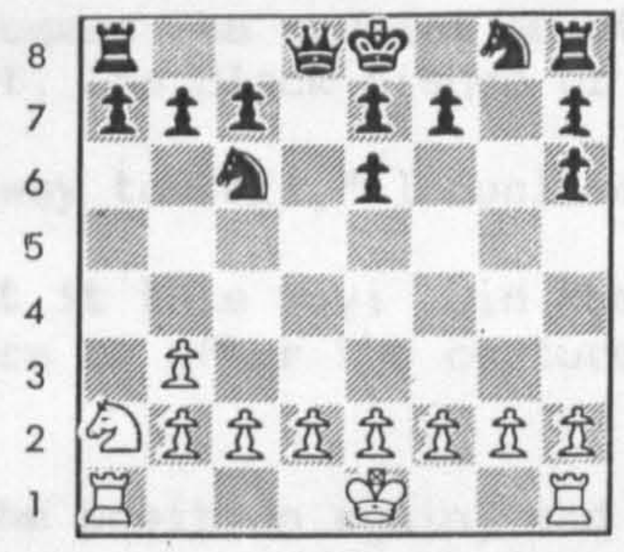
a. Analysing the pieces on board: 'How many pawns are on board?' 'Upon what squares did they capture?'

b. Analysing the pieces not on board: 'What pieces were captured on board?' 'What pieces were captured on their home squares?'

c. Consideration of any initial data: If 'Black can castle' is given, it means that the Black King, at least, has not moved or been in check.

By taking a kind of safari along these trails, in the correct order, and not getting lost in the process, the destination - an answer to the given problem - may eventually be reached. Rules must be obeyed on the safari; usually (and at least in this thesis) these consist of the full rules of chess, but this is not always necessarily so, for problems may be encountered that exist within the framework of a modified set of rules.

The Reader may now care to make a safari of his own, accompanied by Sherlock Holmes, by considering the following problems:



The solution below is presented as a dialogue between Sherlock Holmes and his boon companion, Dr. Watson. Dr. Watson is the commentator.

'The problem is: "On what square was the White queen captured?" (Holmes).

I looked at the position and reasoned thus: "Well, Holmes, I see that White is missing his queen, both bishops, and one knight. Now, two captures can be accounted for by the Black pawns on e6 and h6; the first came from d7 and made a capture on e6, and the second came from g7 and made a capture on h6. Now, neither White bishop ever got out on to the board to be captured by these pawns, since the one from c1 was hemmed in by the pawns on b2 and d2 which have not yet moved, and the one from f1 was hemmed in by the pawns on e2 and g2."

"Good, Watson," interrupted Holmes. "I'm glad you were able to see that yourself."

"Elementary, my dear Holmes," I could not help but jest. "However, I'm afraid my reason cannot carry me much further. All right, we now see that the White pieces captured on e6 and h6 are the queen and

"a knight. So the White queen was captured either on e6 or h6, but I cannot see why the queen could not have been captured on either one of them and the knight on the other."

"Well," said Holmes, "then I'll have to give you some hints in the form of questions. One of the main things in solving these problems is to think of the right questions to ask oneself. Now, what was captured by the White pawn on b3?"

"Obviously a Black bishop," I replied.

"What is the home square of that bishop?"

"Clearly c8, as the bishop from f8 travels only on black squares."

"Right. Now comes the crucial question: Which was captured first, the Black bishop or the White queen?"

"I can see no way to tell," I replied.

"Well then, put it this way: Did the White queen get captured before or after the capture on b3 by the White pawn?"

I looked at the position again, and began to see the point.

"The White queen," I said, "got out on to the board via the square a2, hence the pawn on b3 made its capture first to let the queen out. And, since the pawn captured a bishop, then the bishop was captured before the queen."

"Exactly," said Holmes. "And now, does this not solve the problem?"

"I do not see how."

"Well then, I guess the next question to ask yourself is this: Did the bishop on c8 get captured on b3 before or after the capture on e6?"

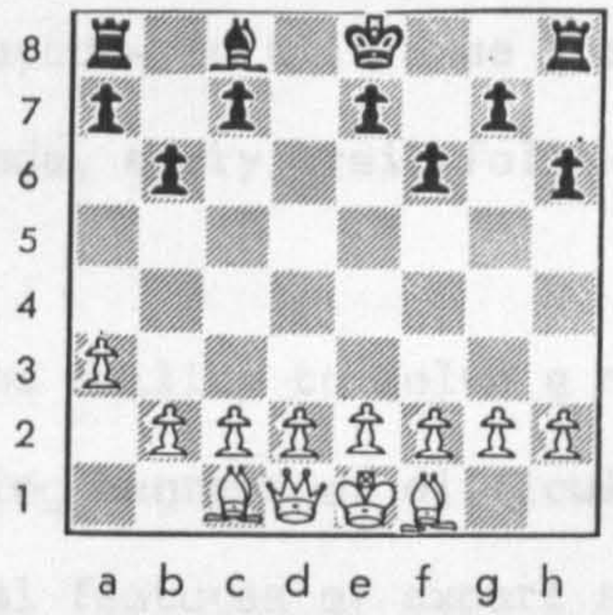
"The pawn on e6 made its capture first," I replied, "to let the bishop out."

"Correct," said Holmes. "Now you have all the pieces of the puzzle, Watson; you have merely to put them together."

"Ah," I said, "now I see it. The capture on e6 was made before the Black bishop got out to be captured on b3, which in turn happened before the White queen got out to be captured. Therefore the White queen was not the piece captured on e6. In other words, the sequence was this: First the knight was captured on e6, then the Black bishop got out and was captured on b3, then the White queen got out, and must have been the piece captured on h6. So the queen was captured on h6."

> "Very good, Watson," said Holmes encouragingly. "I think that with a bit more experience you will be able to do retrograde-analysis."' (Smullyan [2], pp. 18-20)

The Reader will have noticed that in this Socratic dialogue the essence of progressing to a satisfactory conclusion lay in asking the right questions at the right time. It is worth reiterating Holmes' comment: 'One of the main things in solving these problems is to think of the right questions to ask oneself.' This point will be brought out more strongly later. Meanwhile, Holmes sets another problem:



> '"It is Black's move," said Holmes. "Can Black castle?"
>
> Dr. Watson gave the following analysis:
>
> "White's last move was clearly with the pawn. Black's last move must have been to capture the White piece which moved before that. This piece would have to have been a knight, since the rooks could not have got out on to the board. Obviously none of the Black pawns captured the knight, and the Black queen's rook couldn't have captured the knight, because there is no square that the knight could have moved from to get to that position. Likewise the bishop couldn't have captured it, since the only square the knight could have come from is d6, where it would have been checking the king. Hence either the king or the king's rook has made the capture. So, Black can't castle."
>
> "You see," said Holmes, "what progress results from application."' (Smullyan [1], pp. 45-46)

These problems will hopefully have given some idea of the flavour of retrograde-analysis chess problems. A deeper look is given in Chapter 3.

2.0 RETRO - AN OVERVIEW

By way of previous experience or perhaps by perusal of the preceding section (with the helping hand of Mr. Holmes), the Reader will now possess some knowledge of retrograde-analysis, so will be able to understand better the information-processing problem that RETRO is attempting to solve. It is this: Given a basic situation, with perhaps some pertinent initial information, can deductions be made to enable a particular question to be answered? In making deductions rules must be obeyed - in this case the full rules of chess - so that every decision made, every trail followed, must be in accordance with these rules.

RETRO has the ability to solve a number of retrograde-analysis problems of varying degrees of difficulty.

The essential features of expert systems are knowledge representation, control, knowledge acquisition and explanation, and these features of RETRO are discussed below. Other sections in this chapter include a block diagram, an example of RETRO at work, comparisons with other systems, and a challenge to the Reader by Sherlock Holmes (and RETRO!).

2.1 The essential features

Mention has already been made (Section 1.2) of various representations of knowledge, such as production rules and frames, which is the subject of much research. Since the frame concept of Minsky [13] is pertinent to RETRO, it is worthwhile to lay the groundwork by quoting the definition given by Winston [22] of a frame:

> 'A frame is a data-structure for representing a stereotyped situation like being in a certain kind of living-room or going to a child's birthday party. Attached to each frame are several kinds of information. Some of this information is about

'how to use the frame. Some is about what one can expect to happen next. Some is about what to do if these expectations are not confirmed.' (pp. 180-181)

A frame, then, consists of a set of slots and values that specify the expected objects it events:

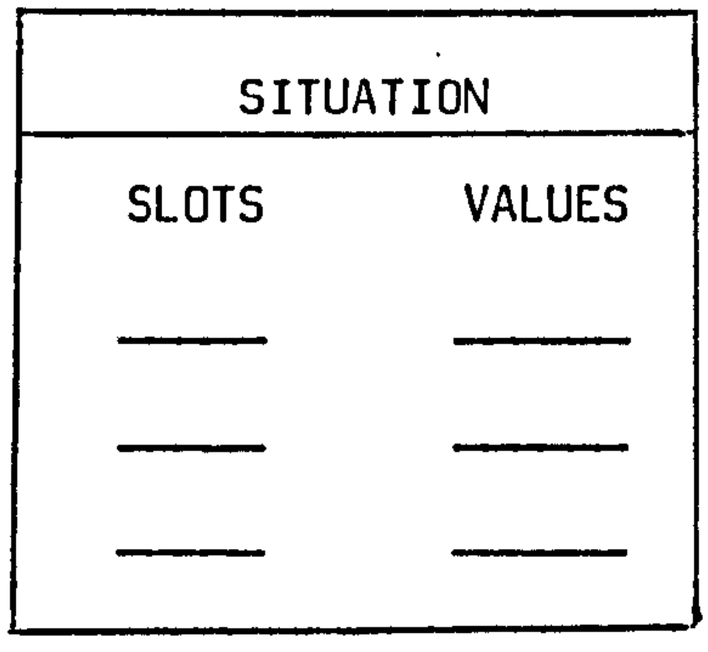


Figure 2.1: Sketch of a frame

What frame systems do is to allow classification of new situations in terms of the stored frame situation, attempting to fill in the slots with values, to determine whether there is a match of the expectations specified by the slots with the new situation.

RETRO uses the frame concept in a different way. The knowledge representation utilised by RETRO is based on the concept of the Significant Event (SE), which is an event of significance that must have occurred at some time in the past history of the game. For instance 'A pawn has promoted' is regarded as an SE, as is 'A king is in check'. The rationale behind this concept is admirably summed up by Sherlock Holmes (Section 1.4):

'One of the main things in solving these problems is to think of the right questions to ask oneself.'

Each SE has questions associated with it: for instance, if the SE is 'A pawn has promoted' then natural questions to ask would be: 'What is the promoted piece?', and 'On what square did the pawn promote?'

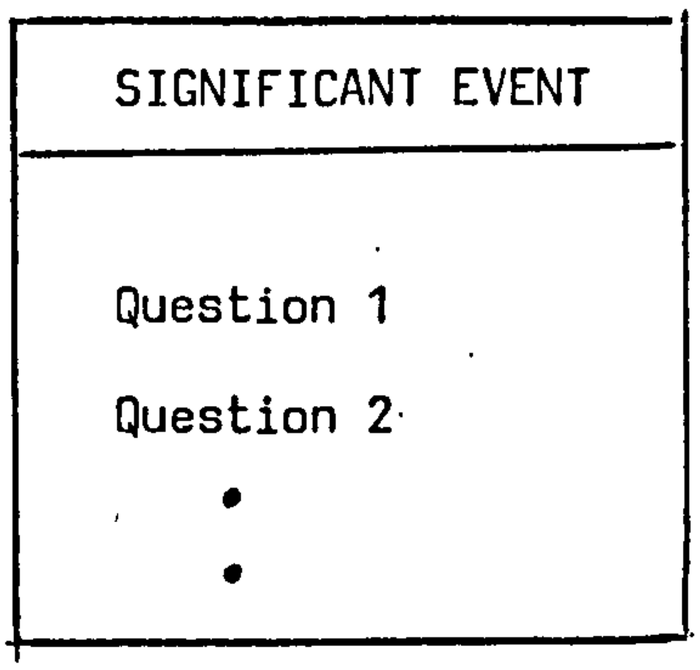


Figure 2.2: Frame used by RETRO

Basically, in solving a problem, RETRO looks to find an SE, and having found one asks the questions associated with it. The answers to these questions may lead to a solution of the problem, or they may point to another SE, with its own associated questions. After each question has been asked there are three possibilities that RETRO considers:

1. A solution to the problem has been found. RETRO answers the given question, then asks if an explanation is required. If the answer is 'Yes' then RETRO prints its chain of its deductions.

2. A solution has not been found. In this case RETRO carries on to the next question.

3. An illegal situation has been encountered. RETRO realises that this line of enquiry is useless.

Control is thus data-driven by the SEs.

The knowledge in RETRO is explicit, so that the system has direct manipulatory access to the knowledge base. It consists of twenty-nine rules, which are split into five groups according to function; such as rules for deducing that a pawn has promoted and rules for checking the legality of reverse moves. The problem of acquiring this knowledge was met by studying many problems. It is not claimed that these rules are exhaustive, merely those that have been encountered so far in the development of RETRO, so it is quite likely that extra

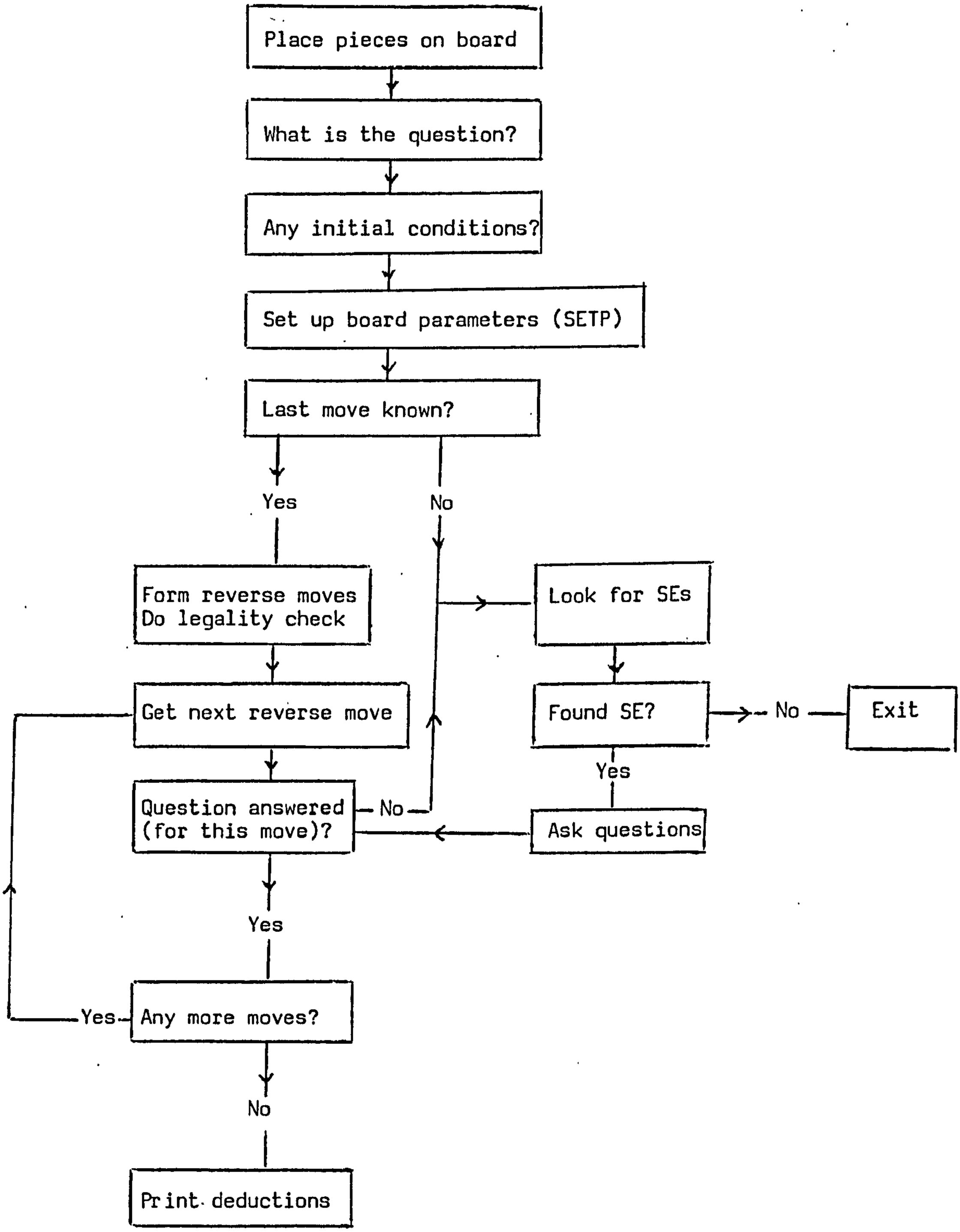
rules will be needed as more problems are considered.

It is unlikely that any expert system will remain in a static condition, but will be subject to modification in the process of time, as extra rules will need to be added and old rules found to be wanting. Ease of modification of the knowledge base is therefore essential, and RETRO has been designed to meet this criterion.

It is becoming axiomatic that an expert system should be able to offer an adequate explanation for its deductive reasoning, for without this facility it is unlikely that it would be acceptable to potential users, particularly if they are highly skilled professionals (less relevant in this domain than, say, medicine, of course, but it still 'feels' desirable). RETRO keeps an ongoing record of its deductions in a dynamic data-base. When an SE is found and when a question is answered then a corresponding entry is made in this data-base, which contains a pointer to a free text explanation of what has happened.

RETRO does not have to disentangle rules to provide its explanations.

2.2 Block Diagram of RETRO



2.3 Functions and variables

RETRO extracts as much information as possible from the given board situation by utilising the function SETP (Set Parameters). This information is then available for use by the SEs and rules throughout the system. If the board is updated in any way - for example, moving a pawn back to its home square - then SETP must be used again to permit RETRO to take cognisance of the situation.

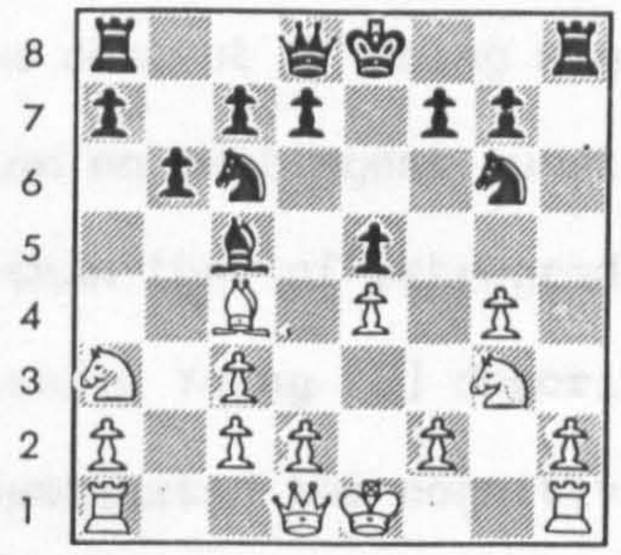
SETP uses other functions as it sets up variables. These are listed below. A list of the more important functions used by RETRO is given in Appendix F.

Function	Returns
BCHSQ	Lists of squares of bishops captured at home
CAPSQ	Lists of squares upon which the pawns must have captured
CPLIST	Lists of pieces missing from the board; i.e. possible black and white pieces captured
CPGTOB	Lists of pieces captured on board; i.e. as CPLIST but less constrained pieces
LFXCAP	Lists of possible pawn cross-captures
MINPWN	Looks at the pawns on board, trying to associate them with definite home squares. It returns the file co-ordinates of squares unaccounted for. These lists are used to determine the possible home squares of promoted pawns. Also returns lists of form [X Y Z] where X = total number of pawn captures Y = number of captures on white squares Z = number of captures on black squares ('Have all pawn captures taken place on one colour?')
POSCAS	Lists of possible 'castle' pieces
NPB/NPW	The number of black and white pawns on board.

SETP sets up the following variables, taking note of any initial conditions such as 'odds given':

Variable Name	
BCAS/WCAS	Lists of possible 'castle' pieces - for example, [[[WR a1][WK e1]][[WR h1][WK e1]]] - used in 'castle' problems: 'Can White castle?', for example. If it is deduced that the WR on h1 must be a promoted piece then obviously White cannot castle on this side so [[WR h1][WK e1]] is deleted from WCAS. POSCAS scans the board for these possible castles.
BLCAP/WHCAP	CPLIST returns these lists of pieces missing from the board; i.e. possible black and white pieces captured. A check is made to see if any piece has been given as odds.
BLEFT/WLEFT	It is possible that a piece was captured in its home square, being unable to leave it. It is also possible that a captured piece was unable to leave its home row. CPGTOB returns the lists of pieces that were able to be captured on board. These lists are used when trying to determine what pieces the pawns may have captured.
BTOT/WTOT	The number of pieces in BLEFT/WLEFT.
BHTAB/WHTAB	MINPWN looks at the pawns on board, trying to associate them with definite home squares. It returns the file co-ordinates of squares unaccounted for. These lists are used to determine the possible home squares of promoted pawns.
BNUM/WNUM	MINPWN also returns these lists, of form [X Y Z] where X = total number of pawn captures Y = number of captures on white squares Z = number of captures on black squares It may be of significance if all pawn captures have taken place on one colour.
BDC/WDC	CAPSQ finds squares upon which pawns must have captured; for example [[[WP h2][g3]]] The White pawn from h2 captured on g3. ('If a pawn made at most N captures, could it have been captured by an opposition pawn?')
XCAPB/XCAPW	LFXCAP looks for possible pawn cross-captures, adding them to BDC/WDC.
CPBX/CPWX	File co-ordinates of the squares upon which pawns definitely captured. Derived from BDC/WDC.
BBHSQ/WBHSQ	BCHSQ returns lists of squares of bishops captured in their home squares.

2.4 RETRO at work - an example



Smullyan [1], p. 105

Question: Is there a promoted piece on board?

Initial Conditions: Both sides can castle.

The Reader may care to attempt a solution of this problem, before Mr. Sherlock Holmes presents his explanation and before RETRO's solution is given.

Mr. Sherlock Holmes elucidates

> 'The piece captured on c3 was not the Black queen's bishop, nor the pawn from h7, since it could not get to the c-file.
>
> Hence the pawn from h7 has promoted.
>
> It promoted on g1 after capturing exactly one piece - the White queen's bishop. The capture by the pawn on c3 had to occur before the bishop got out to be captured by the Black pawn, and so the capture on c3 occurred before the promotion.
>
> Thus the promoted Black piece is still on board.'

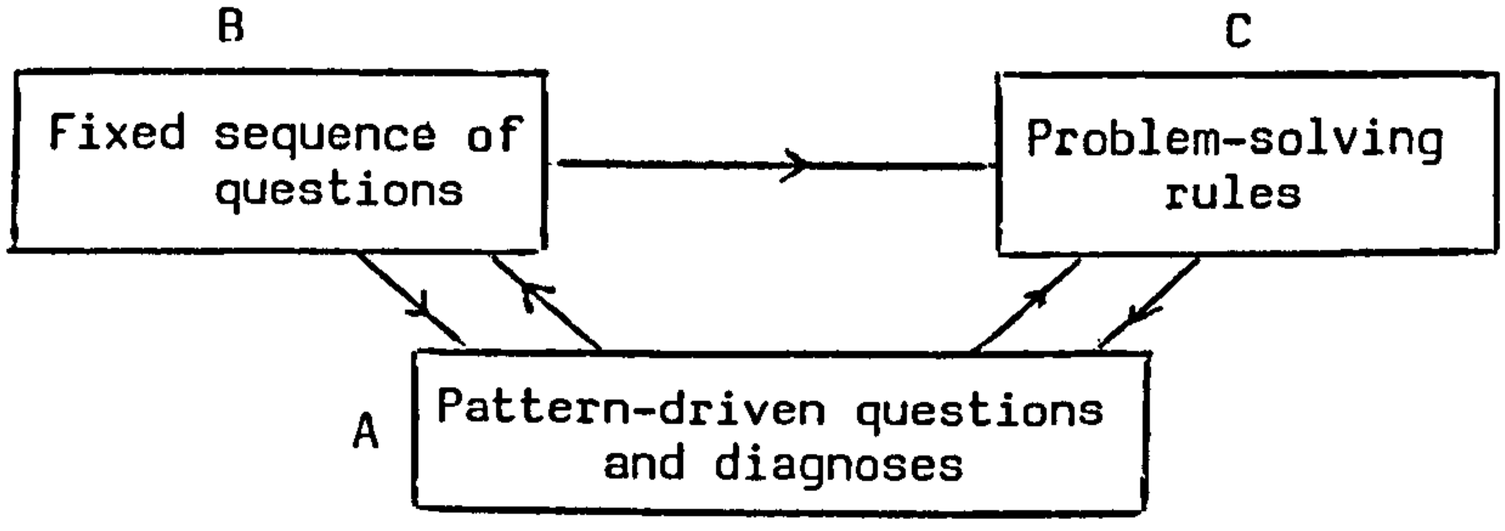
RETRO elucidates

> IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT CAN BE CAPTURED ON BOARD AND THESE OPPOSITION PIECES INCLUDE A PAWN THAT COULD NOT HAVE REACHED A CAPTURE SQUARE THEN A PAWN PROMOTED [RULE 001].
>
> THE BP FROM h7 PROMOTED.
>
> THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES [g1].
>
> IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES [g2].
>
> IF A PAWN CAPTURED ON ITS 1ST MOVE AND COULD ONLY HAVE CAPTURED AN OPPOSITION PAWN AND THE OPPOSITION PAWN MADE N CAPTURES IN PROMOTING, WHERE N IS THE NUMBER OF PIECES CONSTRAINED BY THE CAPTURING PAWN ON ITS HOME SQUARE
>
> THEN THE PAWN CAPTURED BEFORE THE OPPOSING PAWN PROMOTED [RULE 025].

2.5 An analogy

The concept of using significant events or patterns to focus attention on contingent questions appears to have relevance in other fields than that of retrograde-analysis.

Richard Young [8] describes research carried out by John Fox, who investigated the cognitive processes of hospital physicians performing a differential diagnosis of patients suffering from dyspepsia. Three distinct components of the doctor's skill were distinguished, as follows:



Fox found that doctors' actions are often made on a pattern-recognition-like basis, in which a particular question to ask (or tentative diagnosis) is suggested by a particular configuration of symptoms. In other words, a certain pattern will trigger certain questions. This is module A.

However, early in the diagnosis not enough information is available about the patient, who 'presents' just enough information about age and sex, and of course the dyspepsia. Thus the doctor has no alternative but to begin asking questions in the hope that some significant pattern will emerge. There are no grounds for asking these questions differently on one occasion than on another, so questions tend to be asked in a routine order. This is represented by module B.

These two modules (A and B) now begin to co-operate, in that as the questions are asked (in the standard order) information is gradually

accumulated until at some point a rule may be triggered in the pattern-driven module. This will perhaps lead to the asking of a contingent question, which could well trigger another of the pattern-driven rules. If this chain leads to a diagnosis, well and good; otherwise the sequence of questions in module B is continued. Thus the actual path is heavily data-dependent.

Perhaps the above system may fail, the doctor reaching the end of his questions without a diagnosis being reached, in which case he continues in a different manner, considering what diseases are consistent with the evidence so far,and chooses further questions to discriminate between them. This is module C.

The analogy with RETRO is clear. The doctor asks a fixed sequence of questions in the hope of finding some significant pattern.

RETRO looks for Significant Events.

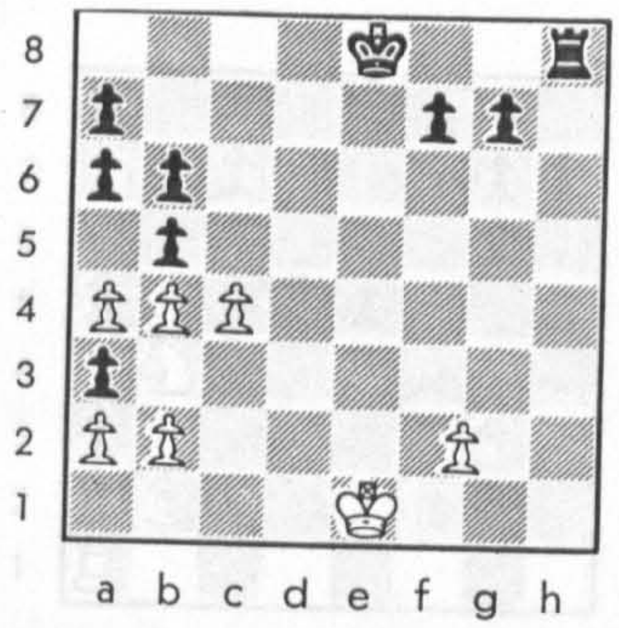
In each case the finding of a significant pattern or event will cause contingent questions to be asked that may or may not lead to an answer.

2.6 A challenge to the Reader

During the course of this work a number of retrograde-analysis problems will be encountered, these problems serving both as illustrations to the text and as examples of the way RETRO works. A small selection of these problems is presented below, together with RETRO input and RETRO answer to each problem but without explanations.

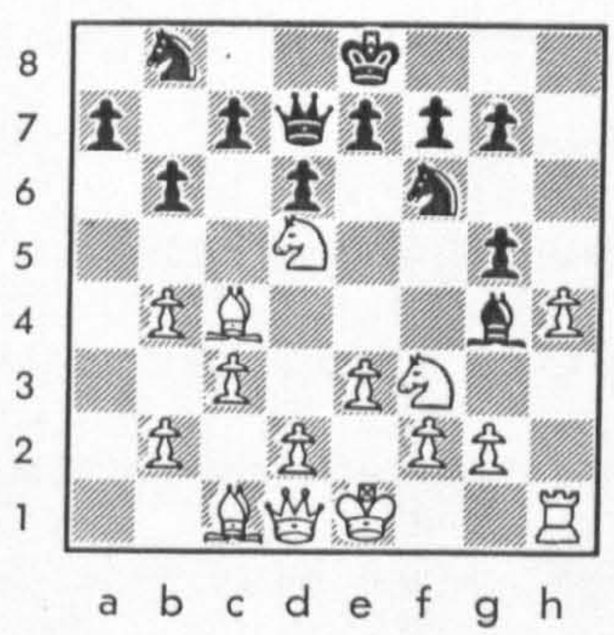
The Reader may care to analyse some or all of these problems to see if his conclusion is the same as that of RETRO. The explanations of the problems will be met in later sections (5.1 and 7.0).

1.



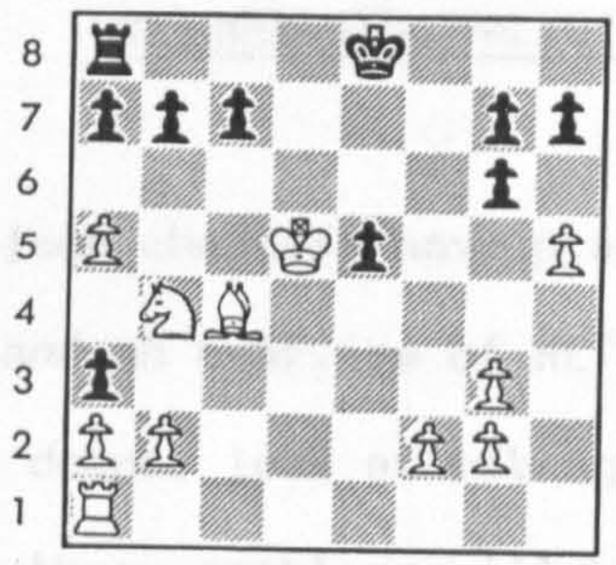
```
*    WHAT IS THE QUESTION
:    [IS THE WP ON f2 OR g2?]

*    ANY INITIAL CONDITIONS (Y OR N):  Y
*    PLEASE ENTER
:    [BLACK CAN CASTLE]

*    ANY MORE CONDITIONS (Y OR N):  N

*    WP IS ON f2.
```

2.



```
*    WHAT IS THE QUESTION
:    [CAN WHITE CASTLE?]

*    ANY INITIAL CONDITIONS (Y OR N):  N

*    WHITE CANNOT CASTLE.
```

3.



a b c d e f g h

```
*   WHAT IS THE QUESTION
:   [CAN BLACK CASTLE?]

*   ANY INITIAL CONDITIONS (Y OR N):  Y
*   PLEASE ENTER
:   [BLACK MOVED LAST]

*   ANY MORE CONDITIONS (Y OR N):  N

*   BLACK CANNOT CASTLE.
```

3.0 RETROGRADE-ANALYSIS - A DEEPER LOOK

Previous chapters have presented a brief introduction to retrograde-analysis and an overview of RETRO. Before examining RETRO in more detail, a deeper look at retrograde-analysis is indicated so that the nature of these problems will become clearer.

Retrograde-analysis problems, unlike conventional chess problems (with requirements such as 'White to play and mate in three'), are concerned only with the past history of a position, the normal look-ahead and search of future moves being replaced by 'look-back', where all possible reverse moves, or sequence of events, may have to be considered. A feature of this look-back is that no emphasis is placed upon the 'best' reverse move, but that all possible reverse moves, no matter how seemingly bizarre, must be considered as having equal probability of occurrence. It is common, for instance, for a pawn to promote not to a queen, but to some other piece.

The general form that such problems take is that a position is given, perhaps with some attached conditions (for example, that neither side made a capture on its last move), the task then being to answer some question such as 'Have any pawn promotions taken place?' or 'Can White castle?'. No restrictions are placed upon the board situation presented or type of question posed; these are dependent only upon the skill of the composer. For most problems the full rules of chess apply, although there are those which make use of a modified set of rules. An important form of this latter kind is the Monochromatic problem, where no piece which starts on a light square may ever subsequently move to a dark square, and vice versa. (Thus, for example, a knight may never move and queen-side castling is impossible.) This subset is small, and has been ignored; it would, for instance,

require its own reverse move generator. Sherlock Holmes was in the habit of teasing his friends with problems involving board orientation, no clear indication being given of which direction either side was playing. This type of question has also been ignored.

Those people addicted to problem-solving but possessing only minimal chess knowledge may take heart that very little chess knowledge is required to solve retrograde-analysis problems - little more, in fact, than knowing how the pieces move. Considerable skill in deductive reasoning, however, is required, which is why they might be said to lie on the borderline between logic and chess.

Mention has been made of the book by Raymond Smullyan [1] as a valuable source book of problems; an earlier collection of problems is Dawson and Hunsdorfer [3].

In solving retrograde-analysis problems the activity of a human solver can be looked at as performing a goal-directed search of a large graph-structure, where each node corresponds to a state of partial knowledge about the problem (with the root node representing the original position and any associated initial conditions given) and each one represents the posing and answering of a question (making use of all the facts known at the time when the question is asked). For the human solver the selection of appropriate questions (consciously or unconsciously) is of utmost importance, the solver posing a sequence of questions in turn, in the style of a Socratic dialogue, until the problem is solved. In general, the answer to a question may introduce one or more constraints which affect the answers to subsequent ones. Thus, asking two questions in the reverse order will not invariably give the same result, and over a series of questions the order will generally be of crucial importance.

Any program written to solve retrograde-analysis problems must take

account of (a) the question to be answered, (b) initial conditions (if any), and (c) the board situation, using this data to find its way to the desired goal of answering the question.

In the following sections examples are given of the type of questions that may be asked, initial conditions that may be given, together with a discussion of the ways in which problems may be classified.

3.1 What sort of questions are asked?

As already stated, the only restrictions upon retrograde-analysis problems are the skill and ingenuity of the formulator. The following table, gleaned from Smullyan [1], gives some idea of the range and scope of the questions that may be asked:

On what square was the White queen captured?
Prove that a promotion has taken place.
What is the missing piece on h4?
Can Black castle?
White can castle. Can he castle either side?
Is the White pawn on g2 or h2?
A White pawn has been knocked off the board. On what square does it stand?
White has just moved a pawn to f4. Did it come from f2, f3 or g3?
Deduce that there is a promoted piece on board.
On what square stands the White king?
On what square was the other White bishop captured?
On h6 lies an unknown. Where was it 2 moves ago?
What was the last move?
Is the White queen on d2 original or promoted?
What colour is the pawn on f2?

Obviously the type of question posed may vary considerably in detail, but an analysis of the problems in [1] shows that there are three particularly important types, which in order of popularity are as follows:

(i) Castling problems (e.g. does White still have king-side castling rights?).

(ii) Location problems (e.g. does a certain White pawn stand on square g2 or h2?).

(iii) Missing piece problems (e.g. the piece on square h4 has fallen off the board; which piece was it?).

3.2 What are the initial conditions?

Problems frequently have associated 'initial conditions' which give information about the past history of the game. These too can vary considerably, as the following table shows:

The White king has made under 14 moves.
White moved last.
Black to move. Neither Black nor White captured on his last move.
White gave Black odds of a queen. Both White knights are original.
White to move. No underpromotions.
Black to move. Black can castle.
Neither king has moved.
No promoted pieces on board.
Both sides may castle.
No White pawn has promoted.
White gave Black odds of both knights. Neither king has moved or been in check.
White to move. No captures have been made in the last 4 moves.
Neither king has moved. On f2 stands a Black or a White pawn. A White knight stands on either f3 or f4.

The single most helpful initial condition for the solver is almost certainly to be told which side moved last, because this implies that the first step in solving the problem is to make the reverse move(s) (often there is only one legal reverse move) and look at the consequences. Something of significance is then usually to be found.

3.3 How RETRO 'understands' the question

Now that some indication has been given of the scope of the questions that may be asked (and their initial conditions), it would be pertinent to describe how RETRO 'understands' what it is supposed to do. In order to avoid any immediate misconception it is pointed out that the use of the word 'understand' does not imply that RETRO, in any way, implements some, or any, parts of understanding. It is also not a case

of using wishful mnemonics or the simple-mindedness so aptly described by McDermott [4]. All 'understands' means here is that RETRO must somehow examine the given question and initial conditions (if any), then determine what to do about them.

RETRO performs a simple 'keyword' parsing of the question and conditions, using functions PARS and PARSEC (Appendix G). Having extracted these keywords RETRO then sets certain variables, as the examples in the following tables demonstrate:

Question	Keywords	Action taken by RETRO
What were the last 4 moves?	'last' 'moves'	Find integer 4 Assign 4 to CP where CP is the number of reverse moves that have to be considered
Can Black castle?	'Black' 'castle'	BCAS contains a list of possible castle pieces Set BCFLG = length (BCAS); i.e. BCFLG contains the number of positions that must be considered
Is there a promoted piece on board?	'promoted' 'board'	Set PPONB = 1 This tells RETRO that it is looking for a promoted piece
What is the missing piece on square h4?	'missing' 'square'	Set MISPQ = [h4] MISPC = Ø
Is the WP on f2 or h2?	f2, h2 'WP'	RETRO recognises this as a location problem and acts LOCFLG true [[WP f2][WP h2]] is placed in PLOC.

Initial Condition	Keywords	Action taken by RETRO
Black moved last	'Black' 'moved' 'last'	Assign 'B' to LM (If it is not known which side moved last then LM = 'U'.)
No captures permitted for the last 2 moves	'captures'	Find integer 2 and assign to CP Assign 'N' to CAP (It is assumed that captures are permitted, i.e. CAP = 'Y', unless otherwise stated.)
W gave odds of a WQ	'odds'	Assign WQ to ODG (odds given)
Both WN are original	'both' 'original'	Assign [WN WN] to ORP (original pieces on board)
Black can castle	'can' 'castle'	Look at BCAS. If e.g. BCAS = [[[BR a8][Bk e8]][[BR h8][Bk e8]]] then add [Bk e8] to the list in PNM (pieces that have not moved) - nothing may be deduced about the rooks. On the other hand, if BCAS = (for example) [[[BR a8][Bk e8]]] then add both rook and king to PNM.

3.4 Classification of problems

When first meeting something new or novel, something which at first sight appears to possess a wayward or random nature, then some sort of pattern is sought, a classification to make this 'something' more intelligible. Retrograde-analysis is no exception. Due to the potential wide-ranging nature of retrograde problems it is important that some form of classification be sought, in order to aid both human and computer solvers.

There are a number of possible candidates, such as:

a. Whether it is known which side moved last.

b. In terms of the type of question posed.

c. In terms of techniques which must be applied to give a solution. Such techniques include methods of determining:

   (i) which side moved last;

(ii) the last move played;

(iii) whether a player has 'castling-rights';

(iv) a route for a particular man from its 'home' square to its current position;

(v) possible pawn home squares;

(vi) which men have been captured (and on what colour squares);

(vii) the captures made by pawns.

The solution of a complex retrograde-analysis problem will, in general, require the application of several of the above techniques in a carefully chosen order.

These classifications are interesting, but objections may be raised to all of them:

a. is rather too coarse-grained. If it is known which side made the last move, then the next step is to make the appropriate reverse move(s) and study the board situation afresh. If, on the other hand, it is not known which side made the last move, then what happens? How could the solver set about attempting a solution?

b. is perhaps too restrictive. A system could be designed to solve certain types of problem, but would probably be unable to solve or even attempt to solve problems outside its domain.

c. leads to problems of control. How would the control know what techniques to apply - in a particular order - for all types of board situations and questions?

What is needed is some form of classification that will be of general applicability, one that would also (hopefully) point to the trail that must be followed to arrive at a solution.

RETRO is based upon such a classification - it is called a Significant Event.

3.5 Significant Events (SEs)

An SE occurs when it can be deduced (or observed) that something of significance has happened on the board. For example, the deduction

'A pawn has promoted' is deemed to be an SE, as is the observation 'A king is in check'. SEs are used in RETRO to focus attention upon the questions that must be asked in order to effect a solution.

A full discussion of SEs is given in Section 4.2.

The use of SEs in RETRO is akin to the process of problem solving in Mathematics described by Polya [23], which contains a number of heuristics. Establishing that a SE has occurred is analogous to the proving of a mathematical lemma, which then suggests further questions for the mathematician to ask until a further lemma is proved, and so on until the original problem is solved.

4.0 RETRO KNOWLEDGE REPRESENTATION

4.1 Introduction

Knowledge is represented in RETRO by rules and an adaptation of the frame concept [23], these frame-like structures being Significant Events (SEs). The general use of frames is based on the idea of organising the properties of some object or event to form a prototype. The strength of frame systems lies in the fact that those elements that are conventionally present in the description of an object or event are grouped together and may thus be accessed and processed as a unit. Following frame terminology each frame structure contains slots of information associated with it. A slot filler can either be a constant or the name of another frame, or even meta-knowledge about the frame itself; there may, for instance, be general slots containing book-keeping information, control slots about what to do if the prototype is confirmed or disconfirmed, and rule slots for summary rules, etc.

Standard restrictions may be applied, such as 'unit' and 'range', which specify that certain objects are required to be given, together with a range of values that these objects may take. Default values may be attached to the unit and range fillers.

In RETRO, the SE may be regarded as a form of frame structure. It has a name which identifies the concept with which it is concerned. An SE may also be regarded as having slots, but,unlike the traditional frame with its 'unit', 'range' and default values, these slots contain only questions. The questions (which may be added or removed) are questions the answers to which may lead to the solution of a given problem. Thus there is no idea of the 'completeness' inherent in traditional frames.
A listing of all the SEs in RETRO, together with their associated questions, is given in this chapter. Each SE is described, also the significance of

the questions.

The different groups of rules are presented, with examples of each.

4.2 Significant Events

The RETRO knowledge base includes 4 SEs,which are capable of solving a significant number of problems. These SEs are as follows:

1. It is known which side moved last. This SE is extremely helpful and is usually stated in the initial conditions of the problem. The implication for the solver is that the first step towards a solution should be the generation of all legal reverse moves, after which make each reverse move in turn and examine the board situation. (In many of these problems it will be found that there is only one possible reverse move.)

   RETRO has a reverse-move generator, PCMOV, which generates all reverse moves, irrespective of legality. Legality is checked by means of LEGCHK, which applies 'legality' rules to the prospective moves.

2. A pawn has promoted. The deduction that a pawn has promoted prompts a number of questions to be posed: 'On what square did the pawn promote?', for example. A number of rules have been derived to enable the deduction to be made.

3. A king is in check. This is an observation rather than a deduction. RETRO scans the board using ISKCHK to see if a king is in check. If both kings are in check then an 'illegal position' is registered.

4. It is known exactly what pieces the pawns have captured. At first glance this is not a very exciting SE, but if it is known exactly what pieces the pawns have captured this could lead to the determination of the order in which pieces were captured, and the squares on which they were captured.

The deduction, or observation, that an SE has been confirmed prompts questions to be asked. These are discussed in the next section.

4.3 Significant Events - the questions they prompt

a. It is known which side moved last

Although this is a strong SE in the sense that to the experienced retrograde-analysis solver the action to be taken is obvious - generate all legal reverse moves and examine their consequences - it is the weakest in the sense that it does not lead directly to any questions. Instead, each reverse move is made in turn, which leads to one of the other SEs being triggered.

In general, it is a common feature for one SE to point to another; also it is not uncommon for an SE to call itself recursively.

b. A pawn has promoted

This SE prompts a number of questions that RETRO attempts to answer. These questions are not claimed to be exhaustive.

Question 1: On what square did the pawn promote?

Routine WHSQPP returns the following:

1. A list of possible promotion squares.

2. A list of possible promotion capture squares.

3. A list of those squares where the promoting pawn crossed the last but one rank.

The consequences of these lists are as follows:

They may be involved in a possible castle situation; i.e. if a rook or king is now on the promotion square then it must have moved to allow the promotion. If a king, then a castle is not possible; if a rook, then a castle is not possible on this side.

It may be possible to deduce that for every square upon which the pawn could have promoted then an opposing pawn also promoted. This is not an uncommon occurrence. The deduction that one pawn has promoted may lead to the deduction that an opposing pawn has also promoted.

Likewise it may be possible to deduce that for every piece a pawn could have captured on promoting then an opposing pawn also promoted.

One of the squares where the promoting pawn crossed the last but one rank may have placed the opposing king in check. This again could be of significance in a possible castle situation, for if a king has at any time been placed in check then a castle is not possible.

Question 2: Is the promoted piece on board?

Routine ISPPB is used to see if it is possible to deduce that a promoted piece is on the board.

Consequences:

'A given question may be answered. A board situation may have been presented and the question asked: 'Is there a promoted piece on board?'

If it can be deduced that a promoted piece is on board then ask Question 3: 'What is the promoted piece?'

If no deduction can be made then see if it is possible to determine upon what square the promoted piece could be captured. If this is feasible then consider two possibilities:

a. The promoted piece is on board.

b. The promoted piece was captured on a certain square.

Question 3: What is the promoted piece?

A list of possible promoted pieces is returned by routine WISPP.

Consequences:

It may be possible to determine exactly what the promoted piece is. If, for example, the promoted piece turns out to be a White rook, then this rook could be involved in a possible castling situation.

If at this point a solution has not been found RETRO checks to see if it has been determined that a promoted piece is on board. If this is the case then RETRO goes straight to Question 5; otherwise it asks Question 4.

Question 4: The promoted piece is not definitely known to be on board; can we find a square on which it was possibly captured?

The criterion for this is that the opposing pawns must have made only one capture: for example, if a White pawn has promoted then the Black pawns must have made only one capture.

If no square is found then RETRO asks Question 5; otherwise it states that either the promoted piece is on board or was captured on a particular square.

Question 5: For each square upon which the pawn promoted, or for each piece possibly captured on promoting, can it be deduced that an opposition pawn promoted?

RETRO may have determined that the pawn could possibly have promoted upon more than one square. If this is the case then RETRO assumes that the pawn promoted on each square in turn to see if this leads to the conclusion that an opposing pawn must have promoted.

On the other hand, RETRO may have found just one square upon which

the pawn could have promoted, and if this is a capture square then it asks what pieces could have been captured. For each piece in turn RETRO then determines if this leads to the deduction that an opposition pawn must have promoted.

c. A king is in check

Routine ISKCHK scans the board to see if a king is in check. One of the following is returned:

1. Nil, if a king is not in check.

2. The king in check and the checking piece: for example, [Wk WB g1] - the White king is in check from the White bishop on g1.

3. False. This indicates that both kings are in check - an illegal situation.

Consequences:

If a king is in check, find out how this happened. The following routines are used:

CANCHK	Could the checking piece have moved to its checking position other than along the checking line?
MOVLIN	Could a piece have moved from the checking line to uncover check?
MOVCHK	Could a piece have moved from the checking line to uncover check and then be captured by the king?
PROM	Could a pawn have moved from the checking line - to uncover check - and then promoted?

d. It is known exactly what pieces the pawns have captured

This knowledge can determine the order in which pieces were captured, leading to the determination of what piece was captured on what square. This may have indeed been the question, but this SE can also provide useful information, by asking 'What was the home square of the piece captured?'. This could show, for example, that a White rook involved in a possible castle would not be the original White rook; otherwise the

capture sequence could not have occurred.

4.4 Rules

RETRO contains 29 rules, which are grouped according to function. These rules have been derived from a consideration of the problems in Smullyan [1], and are not claimed to be exhaustive or exclusive; it may be that further study of retrograde-analysis problems would reveal additional rules.

The rule groupings are as follows:

1.	Pawn promotion rules (PROMRULE)	8
2.	Legality rules for reverse moves (LEGRULE)	8
3.	'Castling' rules (CASRULE)	7
4.	'Is the promoted piece on board?' rules (IPBRULE)	2
5.	'What is the promoted piece?' rules (WPPRULE)	4

Each rule is stored as a 'triple'; for example, Rule 7, which is a pawn promotion rule, is stored as:

```
RULE007          Category        PROMRULE
RULE007          Premise         [RULE7]
RULE007          Explain         [R7EX]
```

'Category' is the group to which the rule belongs.

'Premise' is the function to be evaluated.

'Explain' is the function which gives an explanation of the rule.

A list of the rule numbers is kept in ALLRULES; i.e.

ALLRULES = [RULE001 RULE002 ...]

When, for instance, the pawn promotion rules are called, the rules are selected one by one from ALLRULES, and if the category is PROMRULE then the premise is evaluated, returning true or false. If 'true' is returned then an entry is made in a dynamic data base (Section 6.2), which is an ongoing record of deductions made by RETRO.

Examples of each type of rule are as follows:

RULE001 PROMRULE

If a pawn captured on promoting and the only piece it could

have captured is an opposition pawn

Then an opposition pawn promoted.

RULE012 LEGRULE

Own king in check and no way found for this to happen.

RULE022 CASRULE

The promoting pawn placed the opposition king in check from a certain square so Black/White cannot castle

Here actual values for the square are printed, and whether it is Black or White is determined.

RULE024 IPBRUL

If 1 pawn captured on its 1st move

and could only have captured an opposition pawn

and the opposition pawn could not have made any captures

and the home square of the capturing pawn is on the opposition pawn promotion file

Then the pawn captured before the opposition pawn promoted.

RULE027 WPPRUL

If a side has more than one bishop travelling on the same colour

Then one of them must be promoted.

Listings of all 29 rules are given in Appendices A-E.

5.0 RETRO CONTROL

The control facility in RETRO lies in the SEs, for at any one time RETRO is processing a single SE. This association of control with SEs provides the following advantages:

It allows domain experts to specify different control tasks for each SE.

The control knowledge is separate from the inference rules.

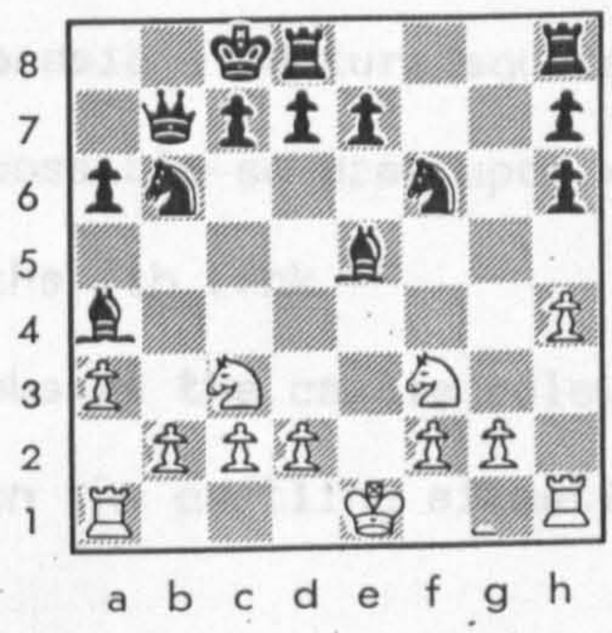
Entire SEs may be added or removed.

Flexibility to define new questions (for the SEs) as required.

The control process is best illustrated by example.

5.1 How RETRO works

1. Smullyan [1], p. 58



Question: Can White castle either side?

Initial conditions: Both White knights are original.

White gave odds of a queen.

The question is scanned by the word-parser PARS, the keywords being 'WHITE' and 'CASTLE'.

WCAS, the list of White castle pieces, contains

[[[WR a1][Wk e1]][[WR h1][Wk e1]]], i.e. 2 possible castle situations, so WCFLG is set to 2.

The initial conditions are now scanned by the word-parser PARSEC, which assigns the list [WN WN] to list ORP (original pieces still on board), and [WQ] is added to list ODG (odds given).

The initial variables are set up by function SETP, as described in Section 2.3, which deletes the pieces given in ODG from the list of pieces that could be captured on board (WLEFT).

RETRO now seeks an SE, using the information that it has been given and derived, and having concluded that a pawn has promoted passes control to this SE. The SE now asks the questions associated with it.

Question 1: On what square did the pawn promote?

This information is derived by function WHSQPP, which returns the following lists:

PSQS	possible promotion squares
LIFC	possible capture squares on promoting
XYLIFS	possible squares upon which the pawn crossed the 7th rank

CHKFLG now looks at the castle rules to see if this information has any bearing upon the castling situation. It does not, so on to the next question.

Question 2: Is the promoted piece on board?

Function ISPPB uses its associated rules but is unable to make a deduction, so to

Question 3: What is the promoted piece?

Function WISPP determines the promoted piece to be a White rook. CHKFLG looks to the castling rules and this time a rule is triggered, so 1 is subtracted from WCFLG (the number of castle positions to consider)

whilst the rook causing the trigger is deleted from WCAS (the list of possible castle candidates).

SE control now asks if it is possible to find the square upon which the promoted piece could be captured by a pawn. The answer is yes [h6].

Now printed is:

'Assuming the promoted piece is on board'

then the deductions made so far,

All flags are reset, then printed is:

'Assuming the promoted piece was captured on h6'.

Control realises that the capturing pawn (of the promoted piece) was captured after promotion, so it makes a reverse move, and resets all variables by SETP.

Question 4: For each square upon which the pawn promoted can it be deduced that an opposition pawn also promoted?

Using the pawn promotion rules, the answer is yes, the Black pawn from b7 promoted.

Question 5: On what square did this pawn promote?

WHSQPP returns a1.

CHKFLG now looks at the castle rules and prints the result of its deductions.

This problem was a favourite of Sherlock Holmes, being composed by his brother Mycroft. As Holmes commented:

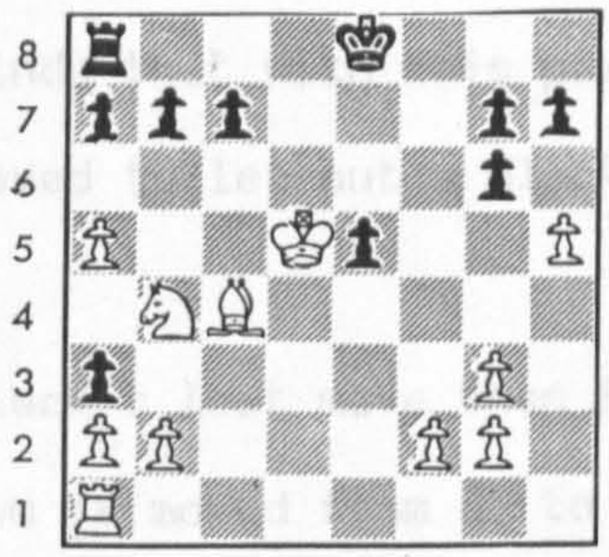
'The interesting thing about this problem is that there is no way of knowing on which side White can castle; all that can be shown is that he cannot castle on both sides.'

The RETRO solution, as printed, is given below:

```
ASSUMING THE PROMOTED PIECE IS ON BOARD
WE CAN POSSIBLY CASTLE WITH THE FOLLOWING
[WR a1]

ASSUMING THE PROMOTED PIECE WAS CAPTURED ON [h6]
WE CAN POSSIBLY CASTLE WITH THE FOLLOWING
[WR h1]
```

```
EXPLANATION REQUIRED (Y OR N):  Y

IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT
CAN BE CAPTURED ON BOARD AND THESE OPPOSITION PIECES INCLUDE
A PAWN THAT COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE001]

THE WP FROM e2 PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g8 f8 e8]

ON PROMOTING IT CAPTURED ON ONE OF THE FOLLOWING SQUARES
[g8 e8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[g7 f7]

IF ALL POSSIBLE PROMOTION PIECES ARE PLACED ON THE
PROMOTING SQUARES
THEN ONLY THE FOLLOWING CAN MOVE

[WN WR]

THE PROMOTED PIECE IS AS FOLLOWS
[WR]

IF THE PROMOTED PIECE IS THE WR ON h1 THEN W CANNOT
CASTLE ON THIS SIDE

THE PROMOTED PIECE IS ON BOARD OR WAS CAPTURED ON ONE OF
THE FOLLOWING SQUARES
[h6]

IF ONE SIDE HAS A BISHOP CAPTURED ON ITS HOME SQUARE AND
THERE IS A BISHOP ON BOARD TRAVELLING ON THE SAME COLOUR
SQUARE THEN A PAWN PROMOTED [RULE004]
IF A PAWN CAPTURED ON PROMOTING AND THE ONLY  PIECE IT
COULD HAVE CAPTURED IS AN OPPOSITION PAWN
THEN AN OPPOSITION PAWN PROMOTED [RULE007]

THE BR FROM b7 PROMOTED

FOR EACH SQUARE ON WHICH THE PAWN MAY HAVE PROMOTED OR
CAPTURED IT HAS BEEN DEDUCED THAT AN OPPOSITION PAWN
ALSO PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[a1]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[a2]

A PAWN PROMOTED ON a1 SO THE WR MUST HAVE MOVED
SO W CANNOT CASTLE ON THIS SIDE
```

2. Smullyan [1], p. 39



Question: Can Black castle?

Initial conditions: Black moved last

The question is scanned by the word-parser PARS, the keywords being 'BLACK' and 'CASTLE'.

BCAS, the list of Black castle pieces, contains [[[BR a8][Bk e8]]], i.e. 1 possible castle situation, so BCFLG is set to 1.

The initial condition is scanned by PARSEC, which sets the variable LM (last move) to 'B'.

RETRO now acts upon the SE 'Black moved last' and generates all possible reverse moves for Black.

RETRO recognises (via BCFLG) that it is dealing with a castling problem for Black so it deletes all reverse moves involving the Black rook and king. (If either of these moves then obviously Black cannot castle.)

A legality check is now applied to the remaining moves, leaving as the only possible moves:

[[BP g6 f7][BP a3 a4]]]

i.e. the only two possible reverse moves for Black are a Black pawn from g6 to f7, and a Black pawn from a3 to a4.

RETRO now considers each move in turn.

a. The Black pawn on g6 is moved to f7 and SETP is run to reset all variables.

CHKFLG is now run. Since a 'castle' flag is set RETRO enters the castle rules and finds that with this pawn on its home square the Black king must have moved to let out a Black rook that was captured on board.

So if this were Black's last move then Black cannot castle.

b. The Black pawn is moved from a3 to a4.

CHKFLG finds no significance in this move, so RETRO looks for an SE. It finds that the White pawn from d2 must have promoted, so now asks the questions associated with this SE.

Question 1: On what square did the pawn promote?

Function WHSQPP returns the information that the pawn promoted on d8 and crossed the 7th rank on d7.

CHKFLG looks through the castle rules and finds that the promoting White pawn must have placed the Black king in check.

So, with this move also, Black cannot castle.

This problem shows how one SE may call another SE, which required only one of its associated questions to be asked before an answer was found.

The printed RETRO solution is given below:

```
BLACK CANNOT CASTLE

EXPLANATION REQUIRED (Y OR N):  Y

'IF THE KING OR ROOK MOVED THEN B CANNOT CASTLE
 THE FOLLOWING MOVES DELETED

 [[BR a8 b8][BR a8 c8][BR a8 d8][Bk e8 d7][Bk e8 d8][Bk e8 e7]
 [Bk e8 f7][Bk e8 f8]]

 INITIAL REVERSE MOVE LIST

 [[BP g6 f7][BP e5 f6][BP e5 d6][BP e5 e6][BP e5 e7][BP a3 a4]]

 PIECE MOVED FROM CHECK POSITION - FOLLOWING DELETED

 [[BP e5 e6]]

 TOO MANY PAWN CAPTURES - FOLLOWING DELETED
```

```
[[BP e5 d6][BP e5 f6]]

NUMBER OF PIECES CAPTURED ON BOARD LESS THAN OPPOSITION PAWN
CAPTURES - FOLLOWING DELETED

[[BP e5 e7]]

REVERSE MOVE LIST

[[BP g6 f7][BP a3 a4]]

BP MOVED FROM g6 TO f7

THE BK ON e8 MOVED TO LET OUT A BR THAT WAS CAPTURED ON BOARD
SO B CANNOT CASTLE

BP MOVED FROM a3 TO a4

IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT CAN BE
CAPTURED ON BOARD AND THESE OPPOSITION PIECES INCLUDE A PAWN
THAT COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE001]

THE WP FROM d2 PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[d8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[d7]

THE PROMOTING WP PLACED THE BK IN CHECK FROM d7 SO
B CANNOT CASTLE
```

5.2 What happens if RETRO is unable to find a solution?

If RETRO, after going through all the SEs in its slots, is unable to find an initial SE, then it prints:

'Unable to find an initial SE'

and exits.

If, on the other hand, it is unable to determine a solution after finding an SE and asking the appropriate questions, then it prints out the contents of its dynamic data-base, with explanations. This will show what deductions it has been able to make.

5.3 Illegal situations

Sometimes an illegal situation is encountered; for instance, it may have been deduced that a pawn captured on promoting, but subsequently it is shown that there was no piece that it could capture. When this happens ILLFLG is set to 1.

It will be recalled that after each question has been asked CHKFLG looks again at the situation. When CHKFLG detects that ILLFLG has been set it realises that an illegal situation has occurred and that this line of questioning will be unproductive. Control is therefore handed back to RETRO.

Illegal situations are implicit in location problems, the selection of the 'wrong' square invariably leading to this condition.

6.0 KNOWLEDGE ACQUISITION AND EXPLANATION

A basic premise of most expert systems is the ability of the system to present explanations for its deductions. Otherwise it is unlikely to be accepted by highly skilled and professional users. The ability to acquire new knowledge and modify existing knowledge is equally important, for knowledge bases are unlikely to remain in a static condition but will require modification in the light of experience.

6.1 Rule grouping

As described in Section 4.4, RETRO groups rules according to function. There are rules for:

1. Deducing that a pawn has promoted - PROMRULE
2. Determining the legality of reverse moves - LEGRULE
3. Determining whether or not a side may castle - CASRULE
4. Determining if the promoted piece is on board - IPBRULE
5. Determining the possible promoted piece - WPPRULE

These rules are listed in Appendices A-E.

Some of these rules are obvious, from the laws of chess, but others have been derived from a consideration of the problems in Smullyan [1]. It is entirely possible that the study of new problems will bring additional rules to light, which will need to be added to their respective groups.

Suppose that a new rule (for example, RULE$^{0}_{}$61) needs to be added to the knowledge base, where RULE$^{0}_{}$61 deals with castling situations, then the following procedure must be carried out:

a. Add 'RULE$^{0}_{}$61' to ALLRULES, where ALLRULES is a list of all the rules used by the system; e.g. [RULE001 RULE002 ...]

b. Add to RULES

RULE061	category	CASRULE
RULE061	premise	RULE061
RULE061	explanation	REX61

where the premise RULE061 is the function that evaluates the rule and REX61 is the free text explanation of the rule.

Thus, when RETRO is looking at a possible castle situation, it scans the list of rules in ALLRULES to select all rules with category CASRULE which are then evaluated.

6.2 The Dynamic Data-Base (DDB)

The Dynamic Data-Base is a list which contains an ongoing record of deductions made. As an example, it could look something like this:

```
[[[WHSQPP][[h1 g1 f1][h1 f1][h2 g2]]][R50EX]][[DPPROM] NIL
[R76EX]][[DPPROM][[[BP g7]]][R52EX]][[PROMRULE][BP g7][R8EX]]
[[PROMRULE][BP g7][R5EX]][[IPPRUL][a6][R64EX]][[CHCK][BB BN BQ BR]
[WCAPEX]][[WHSQPP][[c8 b8][b8][b7 c7]][R50EX]][[DPPROM][[[
WP d4]]][R52EX]][[PROMRULE][WP d4][RIEX]]]
```

Each entry consists of 3 items:

1. The name of the function used or rule triggered;
2. The result of the deduction;
3. The name of the function that has an explanation for what has happened.

The above list would translate as follows (remembering to reverse the list):

```
IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT
CAN BE CAPTURED ON BOARD AND THESE OPPOSITION PIECES INCLUDE
A PAWN THAT COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE001]

THE WP FROM d4 PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[c8 b8]

ON PROMOTING IT CAPTURED ON ONE OF THE FOLLOWING SQUARES
[b8]
```

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[b7 c7]

POSSIBLE CAPTURES BY THE PAWN ARE
[BB BN BQ BR]

THE PROMOTED PIECE IS ON BOARD OR WAS CAPTURED ON ONE OF THE FOLLOWING SQUARES
[a6]

IF ONE SIDE HAS A BISHOP CAPTURED ON ITS HOME SQUARE AND THERE IS A BISHOP ON BOARD TRAVELLING ON THE SAME COLOUR SQUARE THEN A PAWN PROMOTED [RULE004]

IF A PAWN MADE N CAPTURES IN MOVING FROM ITS HOME SQUARE TO A CAPTURE PROMOTION SQUARE AND THE ONLY PIECES AVAILABLE FOR CAPTURE ARE N OPPOSITION PAWNS
THEN AN OPPOSING PAWN PROMOTED [RULE008]

THE BP FROM g7 PROMOTED

FOR EACH SQUARE ON WHICH THE PAWN MAY HAVE PROMOTED OR FOR EACH PIECE IT MAY HAVE CAPTURED ON PROMOTING IT HAS BEEN DEDUCED THAT AN OPPOSITION PAWN ALSO PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[h1 g1 f1]

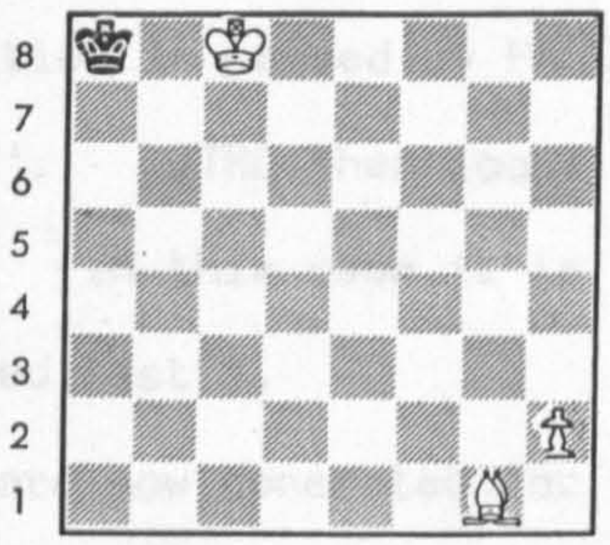
ON PROMOTING IT CAPTURED ON ONE OF THE FOLLOWING SQUARES
[h1 f1]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[h2 f2]

7.0 RETRO - SOME EXAMPLES

In this section a number of problems solvable by RETRO are shown. These problems are of varying degrees of difficulty, involving different SEs. In each case RETRO's solution is presented, followed by a detailed look at the way in which it was found.

Problem 1 (Smullyan [1], p. 16)




```
*   WHAT IS THE QUESTION
:   [WHAT WERE THE LAST 2 MOVES]

*   ANY INITIAL CONDITIONS (Y OR N):  Y
*   PLEASE ENTER
:   [BLACK MOVED LAST]

*   ANY MORE CONDITIONS (Y OR N):  N

*   BK FROM a7 TO a8
*   WN FROM b6 TO a8

*EXPLANATION REQUIRED (Y OR N):  Y

INITIAL REVERSE MOVE LIST
[[Bk a8 b8][Bk a8 b7][Bk a8 a7]]

KING MUST NOT BE ADJACENT TO KING - FOLLOWING DELETED
[[BK a8 b7][Bk a8 b8]]

REVERSE MOVE LIST
[[Bk a8 a7]]

THE BK IS IN CHECK FROM THE WB ON g1

A KING IS IN CHECK, POSSIBLE WAYS IT COULD HAPPEN AS FOLLOWS
[[WN a8 b6]]

BK IN CHECK.   WN MOVED FROM a8 to b6.
```

Explanation

WHAT WERE THE LAST 2 MOVES

RETRO parses this line with the word-parser PARS, the crucial words being 'LAST' and 'MOVES'. Having extracted these RETRO then scans the question for an integer value, in this case 2, which it assigns to variable CP (the number of reverse moves being considered).

BLACK MOVED LAST

This initial condition is parsed by PARSEC, the crucial words being 'MOVED' and 'LAST'. RETRO then looks for either 'BLACK' or 'WHITE', or 'B' or 'W'. In this case it is 'BLACK', which it assigns to variable LM (Who moved last?).

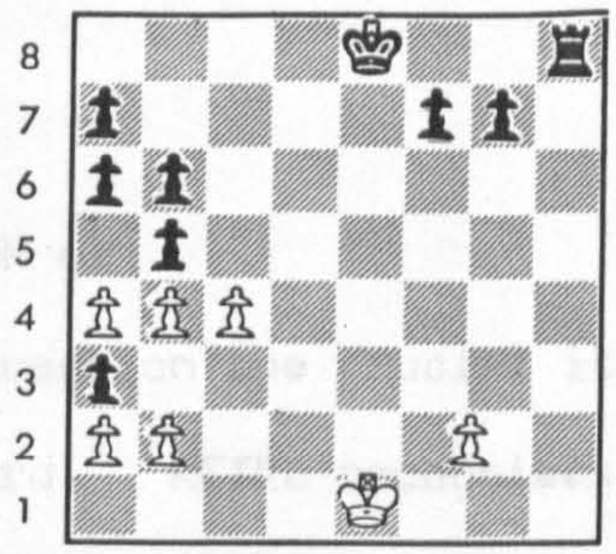
All reverse moves are now generated for BLACK by the reverse move generator PCMOV, and this initial reverse move list is printed. The moves are now subjected to a legality check (Appendix A), two being deleted for the obvious reason that a king may not stand next to its opponent's king.

This leaves one reverse move, which RETRO makes, updating the board.

An SE is now sought, in this case 'A king is in check'. RETRO function KCHECK now looks for possible ways this could have happened, finding that a White knight must have moved from b6 to a8 (being captured by the Black king).

RETRO now checks CP to see if there are any more reverse moves to be found, and,on finding that it has the required number, prints them.

Problem 2 (Smullyan [1], p. 89)



```
* WHAT IS THE QUESTION
: [IS THE WP ON f2 OR g2 ]

* ANY INITIAL CONDITIONS (Y OR N): Y
* PLEASE ENTER
: [BLACK CAN CASTLE]

* ANY MORE CONDITIONS (Y OR N): N

* WP IS ON f2

* EXPLANATION REQUIRED (Y OR N): Y

IF THE PAWNS CAPTURED 8 PIECES AND AN OPPOSITION PAWN
COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE002]

THE WP FROM h2 PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g8]

ON PROMOTING IT CAPTURED ON ONE OF THE FOLLOWING SQUARES
[g8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[h7]

IF A PAWN CAPTURED ON PROMOTING AND THE ONLY PIECE IT
COULD HAVE CAPTURED IS AN OPPOSITION PAWN
THEN AN OPPOSITION PAWN PROMOTED [RULE007]

THE BP FROM h7 PROMOTED

FOR EACH SQUARE ON WHICH THE PAWN MAY HAVE PROMOTED
OR FOR EACH PIECE IT MAY HAVE CAPTURED ON PROMOTING
IT HAS BEEN DEDUCED THAT AN OPPOSITION PAWN ALSO PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g1]
```

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[g2]

Explanation

IS THE WP ON f2 OR g2

In parsing this question the crucial items are f2 and g2, which are squares on the board. RETRO recognises that it has a location problem on its hands so scans the question for the piece under consideration. On finding WP it assigns [WP f2] and [WP g2] to list PLOC (list of possible locations).

BLACK CAN CASTLE

RETRO finds 'CAN' and 'CASTLE' so looks for 'BLACK' or 'WHITE', or 'B' or 'W'. In this case it is 'BLACK', so RETRO then looks at BCAS, which lists the possible castling combinations for BLACK. BCAS here is

[[[BK e8][BR h8]]]

Since only one rook is involved 'BLACK CAN CASTLE' implies that neither the Black king on e8 nor the Black rook on h8 has moved. RETRO therefore adds [BK e8] and [BR h8] to the list PNM (pieces that have not moved).

A feature of location problems is that if the location piece is on an incorrect square then at some time an illegal situation will be encountered. RETRO uses each of the possible location squares to seek this illegal situation, which it tags.

RETRO now seeks an SE, finding by RULE002 that a White pawn has promoted. WHTAB contains a list of possible White promoted pawn home squares, which is used by function WHSQPP to decide from which square the promoted pawn came. The only possible square is h2, so RETRO prints

THE WP FROM h2 PROMOTED

RETRO now asks the questions associated with this SE.

Question 1: On what square did the pawn promote?

Function WHSQPP returns the following:

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g8]

ON PROMOTING IT CAPTURED ON ONE OF THE FOLLOWING SQUARES
[g8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[h7]

Note: The above lists contain only one entry. If there is more than one entry then any subsequent deductions involving the list must consider each square in turn to see if the deduction holds true for each of them.

Question 2: Is the promoted piece on board?

No deductions made.

Question 3: What is the promoted piece?

No deductions as to the actual piece can be made.

Question 4: If the answer to Question 2 is not 'Yes' then RETRO asks if it is possible to find the square upon which it was possibly captured.

No deductions can be made.

Question 5: For each square upon which the pawn promoted can it be deduced that an opposition pawn also promoted?

RULE007 deduces that a Black pawn has promoted, so,utilising BHTAB and WHSQPP, RETRO finds that the only possible home square of the Black pawn is h7.

THE BP FROM h7 PROMOTED

RETRO now returns to:

Question 1: On what square did the pawn promote?

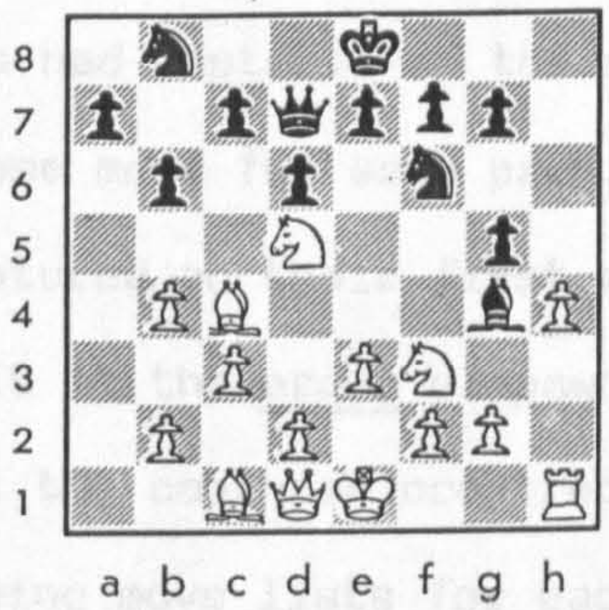
Function WHSQPP returns

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g1]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[g2]

At this point RETRO finds that with the location pawn on g2 an illegal situation occurs, so concludes that the pawn is indeed on f2.

Problem 3 (Smullyan [1], p. 54)




```
*    WHAT IS THE QUESTION
:    [CAN WHITE CASTLE]

*    ANY INITIAL CONDITIONS (Y OR N):  N

*    WHITE CANNOT CASTLE

*    EXPLANATION REQUIRED (Y OR N):  Y

THE WP ON b4 CAPTURED A [BR BR]
THE BP ON g5 CAPTURED A [WR]

A BP CAPTURED THE ORIGINAL WR ON h1

THE PAWNS CAPTURED IN THE FOLLOWING ORDER

THE BP ON g5 CAPTURED A WR FROM h1
THE WP ON b4 CAPTURED A BR
```

Explanation

Parsing the question finds 'CASTLE' as the crucial word. RETRO then looks at the question to discover 'BLACK' or 'WHITE', or 'B' or 'W', here finding 'WHITE'. The list of possible castle pieces is in WCAS, which is

[[[WR h1][WK e1]]]

WCFLG (White 'castle' flag) is set to 1 (length of WCAS)

There are no initial conditions.

RETRO now seeks an SE, finding that in this case it is possible to determine exactly what pieces the pawns have captured; that is, the Black pawn on g5 captured a White rook.

The next move by RETRO is to form lists of the possible pawn moves. If the pawns had captured on their first move then obviously there would only be one move for each pawn, but here it is not known whether the pawns captured on their first move or the second. This is immaterial since it is the order of pawn captures that is required, so RETRO assumes that the captures occurred on the second move and thus forms the following move lists for each side:

[[BP h7 h6][BP h6 g5]]

[[WP a2 a3][WP a3 b4]]

These lists are now subject to a 'sort' in order to determine the sequence of captures, and, if possible, the home square of the piece captured. This is done by making moves in turn to see if the opposing pawn could actually make the capture that it was supposed to make, or was the possible captured piece constrained. In this instance the 'sort' works as follows:

Both pawns are placed on their home squares: the Black pawn on h7 and the White pawn on a2.

The White pawn is moved from a2 to a3.

The Black pawn is moved from h7 to h6.

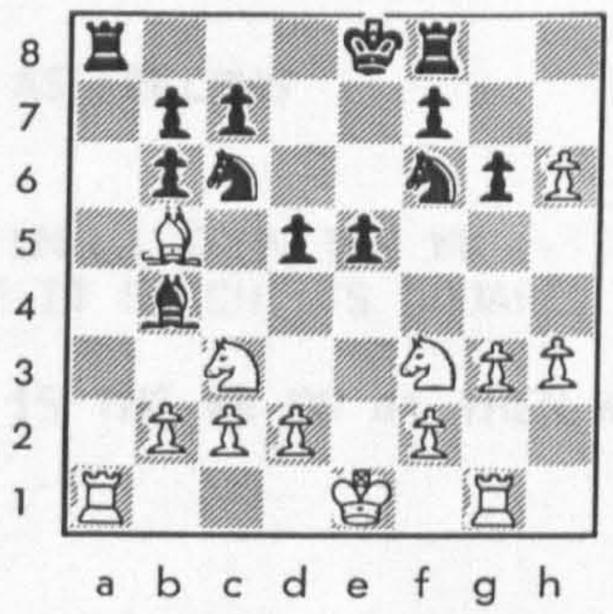
The White pawn is moved from a3 to b4. What could the pawn have captured on b4? Due to the positions of the Black pawns the only possibility is a Black knight, but both Black knights are on board, and since all eight Black pawns are also on board there have been no promotions.

Therefore this sequence of pawn moves is incorrect.

Placing the pawns back on their home squares and moving the Black pawn first leads to the conclusion that the Black pawn must have captured the original White rook from h1 since the rook on a1 was constrained at the time.

CHKFLG is now invoked, which, seeing that a castle flag is set, utilises the castle rules, which determine that White cannot castle since the rook on h1 is not original.

Problem 4 (Smullyan [1], p. 56)




```
*    WHAT IS THE QUESTION
:    [CAN EITHER SIDE CASTLE]

*    ANY INITIAL CONDITIONS (Y OR N):  Y

*    PLEASE ENTER
:    [NEITHER QUEEN HAS MOVED OFF HER OWN COLOUR]

*    ANY MORE CONDITIONS (Y OR N):  N

*    BLACK CANNOT CASTLE
*    WHITE CANNOT CASTLE

*    EXPLANATION REQUIRED (Y OR N):  Y

IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT
CAN BE  CAPTURED ON BOARD AND THESE OPPOSITION PIECES
INCLUDE A PAWN THAT COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE001]

THE WP FROM a2 PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[a8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[a7]
```

A PAWN PROMOTED ON a8 SO THE BR MUST HAVE MOVED
SO B CANNOT CASTLE ON THIS SIDE

IF 1 PAWN CAPTURED ON ITS 1ST MOVE
AND COULD ONLY HAVE CAPTURED AN OPPOSITION PAWN
AND THE OPPOSITION PAWN COULD NOT HAVE MADE ANY CAPTURES
AND THE HOME SQUARE OF THE CAPTURING PAWN IS ON THE OPPOSITION
PAWN PROMOTION FILE
THEN THE PAWN CAPTURED BEFORE THE OPPOSITION PAWN PROMOTED [RULE024]

IF ALL POSSIBLE PROMOTION PIECES ARE PLACED ON THE
PROMOTING SQUARES
THEN ONLY THE FOLLOWING CAN MOVE
[WR]

THE PROMOTED PIECE IS AS FOLLOWS
[WR]

IT THE PROMOTED WR IS ON g1 THEN THE WK
MUST HAVE MOVED TO LET IT REACH ITS SQUARE, SO W CANNOT CASTLE

IF THE PROMOTED PIECE IS THE WR ON a1 THEN W CANNOT CASTLE
ON THIS SIDE

Explanation

In parsing the question by the word-parser PARS, the crucial word is 'CASTLE'. RETRO looks at the question to discover 'B' or 'W', or 'BLACK' or 'WHITE', and not finding any of these assumes that the castling rights of both Black and White are involved.

The lists of possible 'castles' for each side are in BCAS and WCAS, which are, respectively:

[[[BR a8][BK e8]]], [[[WR a1][WK e1]]]

i.e. one possible castle for each side. Therefore both WCFLG and BCFLG are set to 1.

For the initial condition the critical words are 'COLOUR' and 'MOVED'. RETRO then finds 'QUEEN' and 'NEITHER', so adds [WQ BQ] to COLLST, which is a list of pieces that have not moved off their own colour.

RETRO now seeks some SEs, finding that a pawn has promoted.

IF THE PAWNS HAVE CAPTURED ALL OPPOSITION PIECES THAT
CAN BE CAPTURED ON BOARD AND THESE OPPOSITION PIECES
INCLUDE A PAWN THAT COULD NOT HAVE REACHED A CAPTURE SQUARE
THEN A PAWN PROMOTED [RULE001]

As before, RETRO uses WHTAB and WHSQPP, reaching the conclusion that the home square of the promoted pawn is a2.

THE WP FROM a2 PROMOTED

RETRO now asks the questions associated with this SE.

Question 1: On what square did the pawn promote?

Function WHSQPP returns the following:

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[a8]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[a7]

CHKFLG is now consulted, and,since BCFLG/WCFLG have values, RETRO looks to its castling rules with regard to the above information.

A PAWN PROMOTED ON a8 SO THE BR MUST HAVE MOVED SO B
CANNOT CASTLE ON THIS SIDE

BCFLG is reduced by 1, which means that BCFLG is now zero, but since WCFLG has a value RETRO realises it has not finished.

Question 2: Is the promoted piece on board?

RETRO finds that the promoted piece must be on board:

IF 1 PAWN CAPTURED ON ITS 1ST MOVE
AND COULD ONLY HAVE CAPTURED AN OPPOSITION PAWN
AND THE OPPOSITION PAWN COULD NOT HAVE MADE ANY CAPTURES
AND THE HOME SQUARE OF THE CAPTURING PAWN IS ON THE OPPOSITION PAWN PROMOTION FILE
THEN THE PAWN CAPTURED BEFORE THE OPPOSITION PAWN PROMOTED [RULE024]

CHKFLG is again utilised, RETRO looking at its castling rules in response to a value in WCFLG. This time no deductions are made.

Question 3: What is the promoted piece?

RETRO uses function WISPP, finding that if all possible promotion pieces are placed on the promotion square a8 then only a White rook would be able to move.

THE PROMOTED PIECE IS AS FOLLOWS
[WR]

CHKFLG still having a value in WCFLG, RETRO looks at its castling rules, finding that:

> IF THE PROMOTED WR IS ON g1 THEN THE WK
> MUST HAVE MOVED TO LET IT REACH ITS SQUARE, SO W CANNOT CASTLE
>
> IF THE PROMOTED PIECE IS THE WR ON a1 THEN W CANNOT CASTLE ON
> THIS SIDE
>
> WCFLG is now reduced by one, setting it to zero.

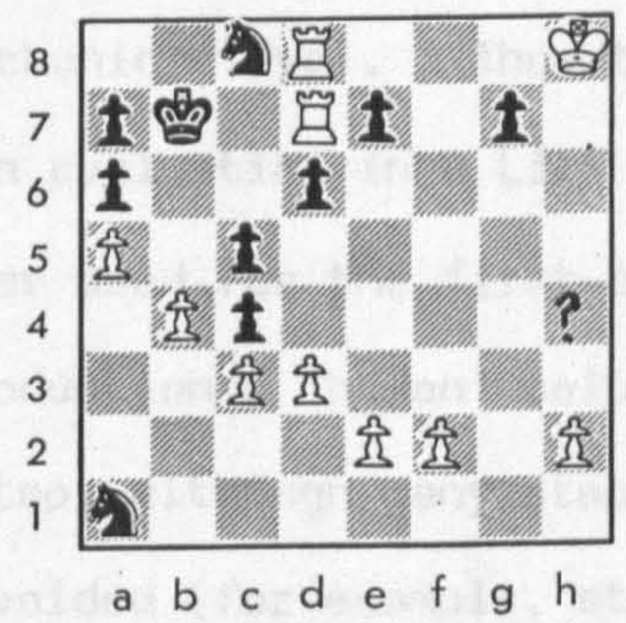
RETRO scans its flags, and on seeing that none is set concludes that it has finished.

8.0 RELATED WORK

Retrograde-analysis, despite having the approval of Sherlock Holmes, does not appear to be a popular field for research. In fact, the only related work that can be determined is the thesis of Robert Filman [10]; this is therefore discussed in some detail. First an overview is presented, followed by a discussion and comparison with RETRO.

8.1 Overview

The aim of the thesis is a consideration of the general issues in the computer representation of knowledge; that is, considering the criteria for a general representation in contrast to first selecting the domain of application and then fitting the knowledge structure. This consideration is based upon the following difficult retrograde-analysis problem (a tour de force for Holmes):



Question: What is the missing piece on h4?

This problem was selected because its solution 'requires' both deductive and observational inferences, where deductive inference is applied in the usual sense, acknowledging, however, that human reasoning proceeds by the immediate recognition of results, 'observation'. Thus, for example, a human player may see that a Black king is in

check. This interaction between deductive and observational inferences is a major emphasis; indeed, the title of the thesis is 'The Interaction of Observation and Inference'.

The representation selected is that of first-order logic, where predicates, functions and constants are declared in this logical system; knowledge being expressed as axioms of the logic. The proof of the above problem lies within these axioms and is meant to be an examination of the reasoning that would be involved in the solution of retrograde-analysis problems. This representation system seeks deductions from these 'first principles'.

In introducing his observational facility Filman dwells upon the strengths and weaknesses of declarative and procedural knowledge before attempting a synthesis of the two, by means of predicate calculus (declarative) plus a means for evaluating the values of predicates and functions (procedural). This semantic procedural attachment is dubbed 'The Chess Eye', to which analogies are given; for example, 'The Mechanic's Eye'. Thus the observation side is performed by function evaluation in a LISP model structure.

The proof checker used for the first-order logic is FOL [16].

After this introduction a 'human' solution to the problem is presented, step by step, although many steps have been included that a human would have avoided (for example, statements to exclude kings in certain situations). This solution is offered as a comparison to the FOL proof.

Following this presentation of the problem together with its representation formalism, considerable space is devoted to developing the chess axioms with their associated chess lemmas and theorems. Objects of the chess world are defined (with FOL declarations) and the rules of chess are expressed with these defined objects, the aim

being to develop general chess theorems from the axioms, not theorems applicable to one small set of problems.

The FOL solution to the problem is then presented. The assertion is made that this proof corresponds in grosser level to the human solution, there being a correlation between the chunks of lines in the proof and the individual steps of the human solution, thus implying the ability of the system to model the human ability to accept problem solutions.

In conclusion Filman stresses that he has not presented a program which would in any sense model the way the human intelligence arrives at the proof. He (Filman) says:

> 'We are interested in the nature of things than an artificially intelligent program would need to be able to do, without specifying the mechanisms by which the program would tie things together.'

The thesis may be regarded as part of the search for epistemologically effective representation formalisms; that is, finding a representation of manageable magnitude that expresses everything we wish to say.

Finally, several other problems are presented, to see how they would look in the formalism, but the proofs are not detailed.

8.2 Critical Review

The above outline (albeit brief) of the thesis of Robert Filman presented some idea of its aims and objectives. The remainder of this chapter consists of a review of its more salient features, with particular reference to RETRO.

8.2.1 Knowledge representation

Different domains of application may influence the choice of knowledge representation, although Aikins [6] has suggested that it is not the kind of knowledge structure that is critical but that the

system should have direct manipulatory access to the knowledge as opposed to having the knowledge built-in. Whatever the representation, however, consideration must be given to the way in which the system utilises this representation; in other words, how efficient is it? Consideration must also be given to the effects of increasing knowledge upon efficiency - does the representation admit of an optimum amount of knowledge before its efficiency becomes unacceptable? The ideal aim is a highly efficient general representation,but the demands of generality could lead to a degradation of efficiency, whilst specifity could lead to efficiency. These generality demands will tend to make predicate calculus systems inefficient; what, for instance, would be the effect on the running time if the number of facts were doubled? Would the running time be squared?

RETRO, with its system of SEs, directs attention to the correct questions to ask; consequently it is very efficient. The average running time is only a few seconds. The effect of increasing knowledge will undoubtedly lead to a loss of efficiency; if more 'pawn promotion' rules are added then RETRO would take longer to check these rules. The system of focussing attention should, however, keep this loss to a minimum.

Writers of predicate calculus systems face another problem - the size of the system. Filman states that his axioms, together with the proof-checker, are already taxing the available memory of his computer system, implying that any additions would be somewhat hazardous. It would appear that the price of generality includes a certain unwieldiness.

8.2.2 Control

In RETRO the SEs, with their associated questions, decide what facts to look at next, whereas in predicate calculus this is not a

separate problem, for those things which are tried include any facts that might fit in accordance with the rules of logic. There is nothing in the basic principles of predicate calculus corresponding to the human question 'What is most likely to be relevant?'; for the application of any particular step is to establish new facts or establish truth or falsity.

8.2.3 Explanation

RETRO will present an explanation of its deduction process, printing the rules which helped it make a decision, so that each step of the proof will be intelligible to a human, who may agree or disagree. The explanation facility is of the highest importance in expert systems, for no expert will accept an answer unless he can see quite clearly the way in which the problem was solved. Feigenbaum [12] states that 'a central organising principle in the design of knowledge-based intelligent agents is the maintenance of a line-of-reasoning that is comprehensible to the domain specialist. This principle is, of course, not a logical necessity, but seems to us to be an engineering principle of major importance'.

RETRO maintains such a line-of-reasoning. In the system developed by Filman the results of its deliberations must be accepted as an act of faith, its very nature preventing an explanation of why it made a particular step.

8.2.4 Illegal positions

The prospective solver of retrograde-analysis problems must always be prepared to meet an illegal situation. This could occur due to a faulty line-of-reasoning or checking the legality of reverse moves. With respect to his own system, Filman writes:

> 'It is worthwhile to emphasise that these chess axioms apply only to situations that might arise in a legal game. Just as formal logic is very sensitive to inconsistency, allowing a proof of any WFF from a false premise, so these axioms, when presented with, for example, an impossible board, do not know which of their axioms to doubt, and will permit the proof of any conclusion about that board.'

It would be interesting to see how such a system would cope with location problems ('Is the White bishop on a3 or a4?'), for the essence of these problems is that with the piece placed on the incorrect square an illegal situation occurs. In addition to location problems, other problems exist where impossible situations may occur, Smullyan [1] providing a good example, where it is required to prove that a given board is illegal.

8.2.5 Degradation

What happens if a system is unable to find a solution? A feature of the expert's expertise is a graceful degradation as the limits of knowledge are reached; if the expert is unable to solve a problem he

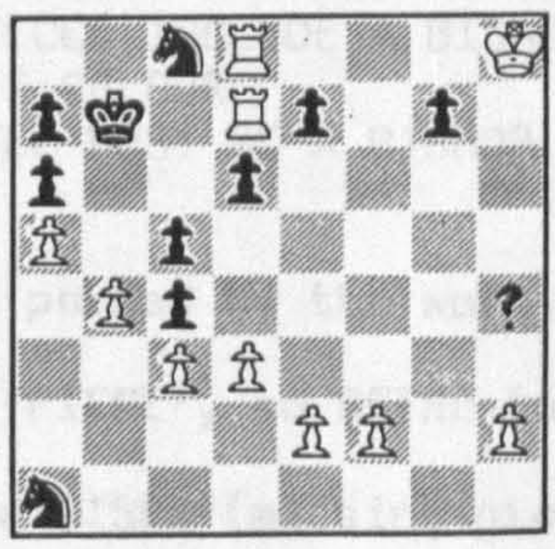
can usually point a finger in the right direction. Expert systems are generally regarded as all-knowing and all-seeing, inevitably arriving at an answer - it may not be the correct one - but an answer there will be. Is this to be expected always? Is it wise to expect it always? Perhaps it would be better if a system were to say: 'Look, I am sure up to this point that I am correct, but not so sure what follows. I have pointed my finger in the right direction; you take it from here.'

A feature of RETRO is that it does degrade gracefully; it may not be able to solve the problem but it will print the result of its deductions, those which it has deduced as correct. So it does give a nod towards the direction to follow. Filman introduces other problems to examine how they would look in his formalism but they would probably be difficult to implement, both for the reasons stated earlier and the fact that his current axiomatisation is oriented towards unknowns centred around particular squares.

8.3 A difficult problem

Filman has developed his formalism around a difficult retrograde chess problem. This is RETRO's solution to the problem:

Smullyan [1], p. 30



```
*    WHAT IS THE QUESTION
:    [WHAT IS THE MISSING PIECE ON h4]

*    ANY  INITIAL CONDITIONS (Y OR N):  N

*    THE MISSING PIECE IS A WB

*    EXPLANATION REQUIRED (Y OR N):  Y

THE BK HAS BEEN PLACED IN CHECK BY THE WR ON d7
THE KING WAS PLACED IN CHECK BY THE WP MOVING
FROM c7 TO d8

A PAWN HAS THEREFORE PROMOTED

POSSIBLE CAPTURES BY THE PAWN ARE
[BB BN]

IF ONE SIDE HAS A BISHOP CAPTURED ON ITS HOME SQUARE AND
THERE IS A BISHOP TRAVELLING ON THE SAME COLOUR SQUARE
THEN A PAWN PROMOTED [RULE004]

THE BP FROM h7 PROMOTED

IF ONE SIDE HAS >2 BISHOPS OR >2 ROOKS OR >2 KNIGHTS
OR >1 QUEEN ON THE BOARD THEN A PAWN PROMOTED [RULE003]

THE BP FROM h7 PROMOTED

FOR EACH SQUARE ON WHICH THE PAWN MAY HAVE PROMOTED
OR FOR EACH PIECE IT MAY HAVE CAPTURED ON PROMOTING
IT HAS BEEN DEDUCED THAT AN OPPOSITION PAWN ALSO PROMOTED

THE PAWN PROMOTED ON ONE OF THE FOLLOWING SQUARES
[g1]

IT CROSSED THE 2ND/7TH RANK ON ONE OF THE FOLLOWING SQUARES
[g2]

IF ALL PAWN CAPTURES OCCURRED ON ONE COLOUR
AND THESE CAPTURES ACCOUNT FOR ALL MISSING PIECES
AND THESE MISSING PIECES INCLUDE A BISHOP THAT COULD NOT HAVE
BEEN CAPTURED ON THIS COLOUR
THEN THE MISSING PIECE MUST BE A BISHOP [RULE030]
```

The question is parsed by the word-parser PARS, the crucial words being 'MISSING' and 'PIECE', so RETRO looks for the square h4, which it places in variable MISPQ (missing piece square), and sets flag MISPC (look for missing piece).

RETRO now seeks an SE, and upon finding that a king is in check

uses KCHECK to determine how this check could have occurred. KCHECK finds that the only way for this check to happen is by a White pawn moving from c7 to d8.

RETRO now says that a pawn has promoted and, since it is known in this case upon what square the promotion occurred, also where the pawn crossed the 7th rank, RETRO has no need to ask the initial questions associated with the SE. It therefore goes straight to the question:

> 'For each square on which the pawn may have promoted or for each piece it may have captured in promoting, can it be deduced that an opposition pawn promoted?'

RETRO has used function PROMCP to determine what pieces the promoting pawn could possibly have captured, finding these to be a Black knight or Black bishop.

For each of these pieces, using its pawn promotion rules, RETRO finds that the Black pawn from h7 must have promoted. For this SE it asks the initial question:

'Upon which square did the pawn promote?'

finding this to be [g1].

CHKFLG now comes into operation, looks amongst its flags, and, upon finding MISPC set, asks function LKFMP to see if enough has been deduced to admit of a solution. LKFMP finds that the missing piece must be White and a bishop.

The SE approach is an heuristic one and thus suffers from problems of incompleteness. The FOL approach is related to the use of a uniform proof procedure to prove theorems in an axiomatic system. Such work is described in detail by Bundy [24]. This latter approach, whilst complete, suffers from the combinatorial explosion and may not produce a solution in finite time.

The difference between the two methods can be exemplified in the context of performing integrals in Mathematics. Those who have struggled

with a variety of integrals will recall that a certain type of integral will suggest an appropriate substitution that will lead to a solution. The uniform solution approach might try all possible substitutions (from a known and sufficient repertoire) in turn, whilst the heuristic approach would recognise the form of a particular type of integral. Naturally this latter approach would fail if unable to recognise the form of an integral, but when successful would give a solution much more rapidly than the former method.

9.0 DISCUSSION

9.1 Representation

The central issue of any expert system is knowledge, how it is represented, utilised and acquired. Representation is concerned with the most effective way or ways of representing the particular domain knowledge - and a domain may require more than one type of representation. Utilisation looks at the design of the inference engine, taking into account both search - the means by which the system works its way to its goal - and control - how it selects among its internal methods. Finally, how is this knowledge acquired - automatically, or after intensive work by a knowledge engineer?

To the above we may add Explanation; an adequate explanation of its deductive processes, with perhaps facility for interrogation by an expert, is an essential part of any expert system.

Our ideal representation should be able to deal with knowledge that is often inexact, incomplete and ill-specified. In addition, it should be tolerant of human foibles when accepting information. Davis [9] also has some pertinent points to make concerning an ideal representation: what, he says, if there is conflicting expert opinion? How could these differences be reconciled in the knowledge representation? Also, is it possible for a system to realise that it is out of its depth and merely say: 'Ask someone else'?. By common agreement a system should degrade gracefully; that is, become less and less proficient as it reaches the limits of its knowledge - as does a human expert. There should certainly not be a calamitous failure.

The following quotation from Sloman[7] succinctly expresses the above:

> 'Work in Artificial Intelligence, whether aimed at modelling human minds or designing smart machines, necessarily includes a study of knowledge. Knowledge about particular domains is the basis of the expertise of all the expert systems described in this volume. General knowledge about how knowledge is acquired, represented and used has to be embodied in flexible systems which can be extended, or which explain their actions. A machine which communicates effectively with a variety of humans will have to use information about what people can be expected to know in various circumstances.'

This work is part of the search for general knowledge about how knowledge is acquired, represented and used.

9.2 Some expert systems

The archetypal expert system is undoubtedly MYCIN [5] - which begat PUFF [13] - which begat CENTAUR [6]. MYCIN has exercised considerable influence upon the development of expert systems, although, like any 'first-generation' system, it is not without failings. Its knowledge lies in production rules which express a uniform 'grain size' of knowledge that is applied in conditions defined by their premises. It is difficult to differentiate between types of rule and there are no expected patterns of data, which may lead to problems of control in that consultation questions may be asked in an unreasonable order; even irrelevant questions may be posed. Another failing is that the acquisition of new knowledge is best done by a domain expert, for a new rule may affect a previous rule, which may affect yet another rule which calls it, and so on.

As an extension of MYCIN [5], a domain-independent version known as EMYCIN [11] has been developed, which in principle can be used as a general purpose framework from which an expert system for any domain can be created merely by adding a specific knowledge base. Van Melle *et al*. [14] have this to say about EMYCIN:

> 'The framework seems well suited for some deductive problems, notably some classes of fault diagnosis, where a large body of input measurements (symptoms, laboratory tests) is available and the solution space of possible diagnoses can be enumerated. It is less well suited for "formation" problems, where the task is to piece together existing structures according to specified constraints to generate a solution.
>
> 'EMYCIN was not designed to be a general purpose representation language. It is thus wholly unsuited for some problems. The limitations derive largely from the fact that EMYCIN has chosen one basic, readily understood representation for the knowledge in a domain: production rules applied by a backward-chaining control structure, with facts about the case represented by associative triples. While the rule representation is quite general, this choice of control structure is unsuitable for problems of constraint satisfaction, or those requiring iterative techniques.'

Alvey [15] reports practical difficulties in implementing an expert system in EMYCIN. The order in which rules are applied, for instance, appears to be the order in which they were last edited.

The above failings of MYCIN [5] and PUFF [13] (which is EMYCIN + a pulmonary disease knowledge base) prompted the development of CENTAUR [6], which uses a coarser grain-size of knowledge by utilising a mixture of frames (prototypes) and rules. The benefit of this approach is that it allows the control structure to determine a more general context before searching for detailed information, the knowledge base being organised into sets of general/specific knowledge about each topic. The questions asked are directly related to the hypothesis being considered. The author of CENTAUR [6], Janice Aikins, lists the following desirable features of an expert system:

1. Representation of knowledge as patterns of data typically encountered in the domain.

2. Classification of actual data patterns in terms of prototypical data patterns.

3. Use of data clues to suggest probable directions for future search.

4. Separation of domain expertise to be applied at different stages during processing.

The next section looks at the way in which RETRO conforms to these terms.

9.3 Comparisons with RETRO

RETRO possesses certain similarities to CENTAUR [6], which may be summarised as follows:

(i) SEs are used to guide processing and explain performance;

(ii) Has direct manipulatory knowledge (i.e. not 'built-in');

(iii) SEs provide the context which guide the questions to be asked;

(iv) Easy inspection of the knowledge base;

(v) SEs are triggered by data;

(vi) The control structure is designed to utilise data clues.

There are, however, basic differences between the two programs; for instance, RETRO does not ask questions of the user but must make the best use it can of the information it is given. Another difference is that RETRO does not use an agenda of tasks to be done. CENTAUR [6] uses its agenda (a) to allow it to deal with a task that has failed, and (b) for the system to 'look ahead' to see what tasks are remaining. Failures can occur when the known information does not support any hypothesis strongly enough to outweigh the disconfirming information, or when not enough information is known upon which to base a decision. If either of these cases occurs then CENTAUR [6] applies its fact-residual rules.

RETRO asks questions associated with an SE, which it can either answer or not. An answer to a question may either lead to a solution or not, in which case RETRO looks for further questions or SEs; if it

can find neither then it is unable to solve the given problem.

RETRO also possesses similarities to the way that doctors seek a diagnosis (Section 2.5). At the start the only information that RETRO possesses is the question asked, the board situation, and the initial conditions (if any). The equivalent of the doctor asking initial questions lies in the hands of function SETP (Section 2.3), which gleans as much information as possible about the situation - this information corresponding to a patient's symptoms. Using this information RETRO seeks an SE (a doctor asks questions in the hope that a significant pattern will emerge), and as with the doctor there are no grounds for seeking SEs differently on one occasion than another, so SEs are sought in a routine order. If an SE is found questions are asked, any one of which may lead to solving the problem, or point to another SE. As with the doctor, the pathway is heavily data-dependent.

RETRO thus utilises the prominent features of CENTAUR [6], and also processes information and attacks problems in a similar way to doctors attempting a diagnosis. SEs may be regarded as patterns of data typically encountered in the domain, the classification of which leads to clues suggesting lines of further search.

9.4 Limitations

Is the SE approach valid for all retrograde chess problems or has chance intervened in decreeing that all problems in Smullyan [1], the source book for this thesis, be amenable to this type of solution? There are two main difficulties in answering this question. Firstly, collections of retrograde chess problems are extremely thin on the ground; as mentioned previously, it has only been possible to trace one previous collection [3], which consists mainly of the 'fairy chess' type of problem. Secondly, as has also been mentioned,retrograde

problems are dependent upon the skill of the composer, so it is always possible that problems could be composed that do not use SEs.

In considering the generality of the SE approach, use must be made of whatever evidence is at hand, and fortunately help is forthcoming in that Raymond Smullyan has recently published a second volume of chess problems [2]. Therefore an analysis of these problems is in order, seeking not to prove the SE approach, which is logically impossible, but rather, in the spirit of Karl Popper, seeking disproof.

This analysis is carried out in the next chapter.

10.0 SOME DIFFERENT PROBLEMS

In contrast to his earlier volume of problems, in which we were subject to mystification and elucidation by Sherlock Holmes, aided and abetted by Dr. Watson, not to mention Professor Moriarty, Smullyan's second volume revolves around the 'Arabian Knights', where the central character is Haroun Al Rashid, posing as the White king, with, as Smullyan puts it '... a host of phantoms, genii, magicians, sorcerers, philosophers, beasts, merchants, hermits, enchanted rocks, and other beings'.

The problems abound with invisible kings, invisible castles, disputed castles, purloined treasure, tales of lazy knights, spies and phantom bishops. This latest volume contains fifty problems, which on the whole are more complex than those in the first volume, and since collections of retrograde-analysis problems are rare it is worth looking at them in some detail.

10.1 Overview

The book contains six chapters, which roughly divide the problems into six groups, as follows:

1. 'Squares and pieces'

The problems here are concerned with certain pieces on certain squares, typical questions being:

What is the piece on g4?
On what square stands the Black king?
On what square stands the White rook?
Is the queen on h5 Black or White?

2. 'Promoted pieces'

As may be guessed, the concern here is with problems of promotion:

Did the Black pawn from b7 promote? If so, is the promoted piece still on board?
The White pawn from a7 has promoted. What is the promoted piece?
What White piece is promoted?
Which is the original and which is the promoted White bishop?

3. 'Arabian Knights'

These are tales of knights, lazy knights (not moved), and knights who have changed armour (colour):

Which White knight has not moved?
Is the knight on a1 Black or White?
Which two knights have changed colour?
Did the White knight on d8 capture the Black queen?

4. 'Phantoms'

A phantom is a piece that is invisible. It may be on the board or off the board:

The White bishop from c1 is a phantom bishop. If on the board, where is it? If off the board, where was it captured?
The White bishop from c1 is a phantom bishop. It was not captured on its own square. Where is it, or where was it captured?

5. 'Genii'

A genie is a piece that should not be on the board. There may, for example, be three White knights on the board, yet no White pawn has promoted. (Traditionally, a genie is never a pawn.)

Which White knight should not be there?
Which officer does not belong on the board?
Which of the bishops on h2 and h4 must be unreal for White to win in 2 moves?

6. 'Queens'

The problems here all deal with promoted queens:

Is the White queen a promoted queen?
Which White queen is promoted?

10.2 Some initial conditions

The above section will have given some idea of the type of questions asked, but what about initial conditions? Are these any different from those in Smullyan's first volume? The following list is typical:

None of the Royalty has yet moved or been under attack.
Black can castle.
There are no promoted White pieces on board.
A White knight was given as odds.
Both kings have moved only once.
The Black queen was captured on its own row.
Black to move.
No knight has been captured by a pawn.

These conditions are similar to the previous ones, which suggests that there may be a 'hard core' of basic initial conditions. The raison d'être of initial conditions, of course, is that they impose constraints, without which a solution may not be possible.

10.3 Significant events

How many of these problems are based on SEs? An analysis reveals that our original SEs are in abundance, and if we tabulate the problems where recognition of a first SE is critical to the solution then we arrive at the following table:

A king is in check	8
A pawn has promoted	22
It is possible to determine what pieces the pawns have captured	10
Last move known	3

(It is not claimed that RETRO can solve these problems.)

If 'trick' questions are disregarded there are four problems that do not appear to be amenable to the SE approach, so it is worth looking at these to see if any pattern emerges.

The first problem asks if a Black pawn has promoted, a suitable question, one would have thought, for the group of pawn promotion rules to handle. However, in this case the pawn promotion rules prove inadequate,

for the problem is solved by reductio ad absurdum; that is, assume the pawn has not promoted and show that this assumption leads to an illegal situation. The argument goes something like this:

> Assume the Black pawn did not promote.
> The Black pawn must have been captured by a certain White pawn.
> An original White officer must have been captured before the Black pawn was captured.
> But no White officer was available to be captured.
> The Black pawn must therefore have promoted.

This argument may be expressed in rule form and added to RETRO's list of pawn promotion rules. It is entirely possible, of course, that other rules based on reductio ad absurdum may be found.

The next two problems ask similar questions: (a) Which of the two White bishops is original? (b) Which White rook is not original? In each case the initial conditions state that a White pawn has promoted. The solutions to the two problems lie in asking if Black pawn captures (could be cross-captures) occurred before the White pawn promoted, and, if so, what did they capture? The answers could lead to constraints being imposed such that the positions of pieces at each stage are known, thus being able to solve the problems. This looks as if another question must be added to the SE: 'A pawn has promoted'.

> 'Could opposition pawns have captured before the promotion took place?'
> If so, what did they capture?'

The last question is, as far as RETRO is concerned, the most difficult. It is as follows:

Question:	Has the White knight from g1 moved only once?
Initial conditions:	The White king has never moved. The Black knight from g8 has moved twice. d7 has never been occupied or traversed more than once.

The solution to this problem depends on determining the squares traversed and occupied at different times, thus determining the

positions of pieces at critical stages. This does not appear to fall within an SE framework, although a possible solution is given in Chapter 12.

Cryptarithmetic is another domain in which the concept of Significant Events appears to play part in formulating a solution. This is discussed in the next chapter.

11.0 CRYPTARITHMETIC

Cryptarithmetic addition problems are like ordinary addition sums except that each digit is replaced (consistently) by a letter; for example 123 + 238 = 361 might be changed to ABC + BCD = CEA. The problem is to find the value of each letter. The rules of the game are as follows:

1. Each letter corresponds to a digit from 0 to 9 inclusive.
2. Each letter stands for a different digit.
3. A number never starts with a zero.

Cryptarithmetic problems are similar to those of retrograde analysis in that both are stylised problems, designed to be solved. In the same way that the majority of chess positions are not amenable to retrograde analysis, one could undoubtedly formulate cryptarithmetic problems that either have no solutions or many solutions.

11.1 The problem

In keeping with the spirit of this work, the solution to a particular problem will be presented by means of a dialogue between Mr. Holmes and Dr. Watson, who between them will bring some interesting things to light. At various stages in the dialogue, when a deduction has been made, a table will appear to clarify the latest situation. The problem being considered is

SEND + MORE = MONEY

and the initial table is as follows:

Col.	1	2	3	4	5	Possible values of remaining digits
		S	E	N	D	
	+	M	O	R	E	0,1,2,3,4,5,6,7,8,9
	c1	c2	c3	c4	c5	possible
=	M	O	N	E	Y	constraints

Figure 11.0

On the left-hand side of the table ci represents the carry out of column i into column i-1, the value of each carry ci being either 0 or 1.

The carry c1 will of course be 0.

The top right-hand corner will give remaining possible digit values, whilst the bottom right-hand corner will present any possible constraints on the values that a particular letter may have; for example, it may have been deduced that D must be less than 5.

11.1.1 A dialogue

'"Now then,Watson," said Holmes, relaxing in his favourite chair, and smoking his beloved pipe, "I have explained the basic rules of cryptarithmetic to you, so what do you make of this little problem?" On saying this Holmes scribbled on a piece of paper and passed it across to me. I looked at it and saw the following:

```
  S E N D
+ M O R E
---------
= M O N E Y
```

"You have become quite proficient at retrograde analysis," remarked Holmes, "so apply your powers of deductive reasoning here." I stared at this problem for some few minutes, but nothing sprang to mind. "I'm sorry, Holmes," I said, "but I can't see anything, I really can't."

"Is there nothing in the situation that arrests your attention, Watson?" asked Holmes despairingly, "Remember what I said about each letter representing a unique digit between 0 and 9."

"And look, Watson," Holmes continued, "To give you a little help I will pencil in the carries out c1 to c5. Remember that a particular carry out ci must either be zero or 1."

I looked again, and then it struck me. "Of course, Holmes," I replied, "Now I see it. We know that a number does not start with a zero, therefore M cannot be zero but must equal c2, which means that both M and c2 are 1."

Col.	1	2	3	4	5	Possible values of remaining digits
		S	E	N	D	
	+	M	O	R	E	0,2,3,4,5,6,7,8,9
	O	1	c3	c4	c5	
=	1	O	N	E	Y	

Figure 11.1

"Very good, Watson," said Holmes encouragingly, "Now given that M is 1, what other deductions can you make?"

I looked hard for a while and finally began to see some glimmer of light. "Well, Holmes," I replied, "Looking at column 2 I can see that S + 1 + c3 must be at least 10 in order to justify a carry out of c2 = 1. Also, even if S took its maximum possible value of 9, then S + 1 + c3 is at most 11."

"Therefore," I concluded, "O must be zero or 1, and since M has already taken the value 1, then O must be zero."

Col.	1	2	3	4	5	Possible values of remaining digits
		S	E	N	D	
	+	1	O	R	E	2,3,4,5,6,7.8,9
	O	1	c3	c4	c5	S = 8 or S = 9
=	1	O	N	E	Y	

Figure 11.2

"Well done, Watson," congratulated Holmes, "but what do you think the next step should be?"

"I think, Holmes," I replied, "that with a little effort I may be able to deduce the value of S, for we now have that S + 1 + c3 = 10, which means that if c3 = 0 then S = 9 and if c3 = 1 then S must be 8. I must now try to eliminate one of these possibilities."

"From column 3," I continued, "We have E + O + c4 = N or E + O + c4 = 10 + N (depending on whether c3 is O or 1). But the second case is only possible if E is 9, c4 is 1 and N = O ... and N cannot be zero because we already know that the letter O represents zero! This means that the first case must be the right one, which implies that c3 is zero, and so (remembering what I said a few minutes ago) S is 9."

"If we now go back and look at column 3," I continued, exhilarated by my own success, "we have E + c4 = N, and since E and N cannot be the same, it follows that c4 must be 1."

Col.	1	2	3	4	5	Possible values of remaining digits
		9	E	N	D	
	+	1	0	R	E	2,3,4,5,6,7,8
	0	1	0	1	c5	E + 1 = N
=	1	0	N	E	Y	

Figure 11.3

"There are times, Watson, when you quite surprise me," said Holmes. "This really is very good."

"It was elementary, my dear Holmes," I could not help but jest, "but do not be too hasty, for I must confess that I see difficulties ahead."

"If I may be permitted to make a suggestion," replied Holmes, "potential carries are always of significance, and you have not yet determined whether c5 is zero or 1. It may be politic to pursue this line of reasoning."

"Thank you, Holmes, let me see what I can make of it. Now then, if c5 is zero then N + R = 10 + E, since we have determined that c4 is 1. On the other hand, if c5 equals 1 then N + R + 1 = 10 + E."

"Good, good, keep going," muttered Holmes, puffing away furiously.

I thought for a few moments, then it came to me. "Of course," I said, "we know that E + 1 = N so we may substitute for N in the above two equations. If we do this, let me see ... yes, E is eliminated, giving R = 9 for N + R = 10 + E and R = 8 for N + R + 1 = 10 + E. But we have already assigned the digit 9, so R must be 8 and c5 must be 1."

Col.	1	2	3	4	5	Possible values of remaining digits
		9	E	N	D	
	+	1	0	8	E	2,3,4,5,6,7
	0	1	0	1	1	E + 1 = N
=	1	0	N	E	Y	

Figure 11.4

"For a first attempt you really are doing quite well, Watson," congratulated Holmes. "Have you any ideas about completing the problem?"

"Well, Holmes," I replied, "I see that there are four unknowns left, together with a possible six values. I suppose I could try a brute force approach and eventually reach a conclusion, but I would rather take a leaf out of your book in trying a process of elimination."

"Good, Watson," exclaimed Holmes. "I can see that you have learned something from me. Please carry on."

"Looking at E + 1 = N," I mused, "the maximum value that N can have is 7, so this means that E cannot be greater than 6. In fact,E must lie between 2 and 6 whilst N lies between 3 and 7."

I studied the problem for a while longer, but could make no obvious deductions. Taking up pencil and paper I quickly realised that with E = 5 and N = 6 left me with D + 5 = 10 + Y, with both D and Y taking up values from 2, 3, 4 or 7. It does not take a genius to see that D must be 7 and Y must be 2.

Col.	1	2	3	4	5	Possible values of remaining digits
		9	5	6	7	
	+	1	0	8	5	
	0	1	0	1	1	E+1=N 2= <E = <6
	= 1	0	6	5	2	3 =<N = <7

Figure 11.5

"I can see, Watson, that your experience with retrograde analysis has paid dividends," said Holmes.'

11.2 The dialogue - an analysis

The above analysis by Dr. Watson will now be considered in more detail. The object will be to determine if any aspects of the analysis stand out, become significant, so to speak. If this is the case then some generalisation may be possible. For convenience the steps are numbered.

1. The first step was in the observation that the sum was longer than the individual parts, thus implying that M was 1, and since M = 1 then obviously c2 must be 1.

2. From Figure 11.1 the next question is: 'What values are S and O?'

3. The value of O was determined by looking at column 2 and examining the constraints of (S + 1 + c3). It was shown that the value must be either 10 or 11, making O to be zero or 1. However, 1 was already allocated to M, so by the rules of the game O must be zero.

4. Given that O = 0 then from Figure 11.2 follows the observation that E + O = N. Now since each letter has a unique value E cannot possibly equal N, so c4 must be 1, with E + 1 = N.

5. c2 and c4 are both known, but c3 is still to be found. Assume c3 to be 1. Then E + 1 = 10 + N or E = 9 + N, but N cannot be zero as this value has been allocated to O. Therefore c3 must be zero.

6. Since c3 is zero then S must equal 9 (Figure 11.3).

7. The next step is to attempt the determination of other values. This is done by considering c5, the only unknown carry, and its effect on column 4.

   If c5 = 0 then N + R = 10 + E
   If c5 = 1 then N + R + 1 = 10 + E

   Now it is already known from step 4 that E + 1 = N, so E may be eliminated from the above two equations, giving

   If c5 = 0 then R = 9
   If c5 = 1 then R = 8

   But the digit 9 has already been assigned to S, so c5 must be 1 and R must equal 8 (Figure 11.4).

8. It appears that the carries (ci) are extremely significant in determining definite values for the given letters. All carries have now been found, so the next step is to apply any possible constraints to the remaining letters. This is done by considering E + 1 = N and the set of remaining possible values [2, 3, 4, 5, 6, 7].

   The maximum value that N can take is 7, therefore E cannot exceed 6, so the following constraints are obtained:

   2 = <E = <6
   3 = <N = <7

9. Some numerical manipulation is now involved, together with the equation D + E = 10 + Y from column 5. By setting E = 5, then N = 6 and it may be quickly seen that D must be 7, with Y = 2 (Figure 11.5).

11.3 Cryptarithmetic and Significant Events

From the dialogue (11.2) and the above analysis it now becomes possible to determine what part (if any) the concept of Significant Events played in effecting a solution. Several questions that proved to be essential in moving towards a solution may be extracted; these are as follows:

Question 1

'Is the length of the sum greater than the length of any component part?'

This question is analogous to knowing which side moved last in RETRO; a crucial piece of information when available.

The answer to the question is 'Yes', which then triggers another question:

'What does this imply about M and c2?'

The implication of course is that both must have the value 1.

Figure 11.1 shows that two values have now been found in column 2, which prompts the next question.

Question 2

'What are possible values of S and O?'

The only way to answer this question is to consider both possible values of the carry-over c3. This is analogous to the reverse-move generator of RETRO, where all legal reverse moves are generated. O was determined by looking at the constraints imposed on column 2. How to determine S? Once again, look for a carry, this time c4. Column 3 now reads 'E ÷ O = N', but the rules of the game exclude the possibility of any digit being assigned to more than one letter. Thus the only way out of the impasse is for c4 to be 1, with 'E + 1 = N'. The value of the carries c2 and c4 are now known, with c3 still undecided, so as before consider both possible values of c3 and see if this leads to any deductions.

Assume c3 to be 1. Then from column 3 it is obvious that E must be 9 and N must be zero, but zero has already been assigned to 0. Therefore c3 must be zero, with S equal to 9 (from S + 1 + c3 = 10) (Figure 11.3). Figure 11.3 also prompts the next question to be asked.

Question 3

'What can be deduced from column 3?'

Column 3 states that 'E + 0 = N' and since c4 is 1 then E + 1 = N, which gives a constraint.

How best to use this constraint?

The only carry still to be determined is c5, so the previous analysis suggests that it may be politic to consider its possible values.

Assume c5 = 0. Then column 4 reads 'N + R = 10 + E'.

Assume c5 = 1. This time column 4 reads 'N + R + 1 = 10 + E'.

Then using 'E + 1 = N' gives the following:

If c5 = 0 then R = 9

If c5 = 1 then R = 8

But the digit 9 has already been assigned to S, therefore c5 = 1 and R = 8 (Figure 11.4).

At this stage all carries have been found and it looks as if numerical values will have to be tried.

Question 4

'Can any constraints be applied?'

From Figure 11.4 the set of possible remaining values is [2, 3, 4, 5, 6, 7], whilst 'E + 1 = N'.

Therefore the maximum value for N is 7, and that of E is 6.

Similarly, the minimum value for E is 2, and that of N is 3.

So 2 = <E = <6 and 3 = <N = <7.

Using these constraints the solution is obtained by numerical manipulation.

It can be seen that, as far as this particular cryptarithmetic problem

is concerned, events occurred that played a significant part in its solution. These events correspond to the first three questions asked, each of which caused other questions to be asked, in which the roles of the carries ci proved to be of crucial importance. This is analogous to the role of SEs in RETRO, in which the triggering of an event caused questions to be asked.

Questions 2 and 3 may be generalised, so this problem has given rise to three major SEs.

1. The length of the sum is greater than the length of any component part.

2. The value of a carry-out of column is is established. This triggers the question:

   'What are values of all the unknown letters in column i and column i - 1?'

3. The value of a letter in column i is established. This triggers the question:

   'What are values of all the unknown letters in column i of ci and column i - 1?'

These SEs therefore cause questions to be asked, analogous to the SEs in RETRO.

As with pawns in retrograde analysis, the carries in cryptarithmetic seem to be of vital importance. Often in retrograde analysis,when a temporary impasse is reached in the steps of the solution, a good strategy is to try one more reverse move and see what happens. A similar strategy here would be to assume values for an unknown carry and see if this leads to any contradictions.

11.4 Summary

In the particular cryptarithmetic problem studied here, it has been shown that the concept of Significant Events can play a part in arriving at a solution, although many problems would need to be considered to

determine if the concept possesses a general application.

12.0 CONCLUSIONS

In this final chapter an assessment is made of what has been achieved, together with a look towards future developments. First, however, a word about retro-analysis problems: here the concern has been with a given board situation and making deductions therefrom, although it is realised that, starting from the initial position, there are undoubtedly many games that would have led to the same situation. The concern has been only in the paths that pieces may have taken and the order in which events must have happened.

12.1 Review

The major aspects of an expert system are knowledge representation, control, knowledge acquisition and explanation; these are now discussed with particular reference to RETRO.

12.1.1 Knowledge representation

In its knowledge representation lies the heart of any expert system ('In the knowledge lies the power'), this representation varying according to the domain of application, which may in fact require more than one type of representation in order to perform its function efficiently.

The representation used by RETRO is based on the concept of SEs, Significant Events, that have occurred some time in the past, and whose significance is that they provide clues to the answering of questions about the current situation. An SE is akin to a frame, but,instead of slots containing information and expected values, there are a number of pertinent questions prompted by the SE, which may solve the given problem or point to another SE. This concept is capable of solving a number of retrograde problems of varying types and complexities, but

whilst appearing to be of wide general application - at least as far as published programs are concerned - it does not appear to be of universal application, in that problems have been noted in which the concept appears to fail (Section 10.3). For most of these problems it has been possible to suggest modifications to RETRO that would enable RETRO to seek a solution, but one particular problem presented difficulties in that it did not appear to conform to the SE concept. What, if anything, can be done about it?

In answering this question, reference is made to Section 2.5, which presented a discussion about the way in which doctors make a diagnosis, the discussion revolving around three modules, A, B and C, where module A consists of pattern-driven questions and diagnosis, module B is a fixed sequence of questions, and module C consists of problem-solving rules. Similarities were found between modules A and B to the way that RETRO operates, in that for both cases pattern-driven questions are sought. Nothing was said, however, as far as RETRO is concerned, about module C, which comes into operation when a pattern cannot be found, further questions being asked in the light of evidence so far. RETRO may benefit from having the equivalent of module C, which would be able to make suggestions as to what to try next.

12.1.2 Control

The control knowledge for RETRO is based upon the questions asked by each SE, and is thus represented explicitly, as distinct from being represented implicitly in inferential knowledge.

After each deduction RETRO checks to determine if this is sufficient to answer the given question, in the ways described earlier. A feature of retrograde chess problems is that an illegal situation is always likely

to be encountered (usually in 'location' type problems), so any program dealing with these problems must have a method of coping with illegality. RETRO is able to detect such a situation, thus indicating that it is on the wrong path and must therefore start again.

In addition, it is important to compare the deductive processes of RETRO with that of a human expert, for a program that more closely parallels human reasoning is more likely to be accepted. Although it is not claimed that the control structure represents exactly the way that human experts reason, it does appear to possess a certain validity in that it parallels the way that physicians reach a diagnosis.

The question of control is of course bound up with RETRO's understanding what it is trying to do and possessing the means of knowing when it has completed its task. It has been pointed out that retrograde problems are dependent upon the skill and ingenuity of the composer, so RETRO is always likely to come across a question, together with its initial conditions, not so far encountered. A way to overcome this would be to develop RETRO's parsing process.

A feature of human expertise is that the expert, when faced with an apparently insoluble problem, usually has enough knowledge and experience to make some deductions from the situation, thus enabling him to give a nod towards the direction in which one must go to seek a solution. In effect, the human expert will degrade gracefully. An expert system must be capable of the same. RETRO is able to exploit its SE capability to make deductions in order to effect this degradation.

12.1.3 Knowledge acquisition

It has been pointed out that the way in which knowledge may be represented has been the subject of much research. If it is accepted that a particular expert system is unlikely to remain static, but will require regular changes to accommodate new knowledge or modify old

knowledge, then the ease and simplicity with which these changes can be made become a critical factor. Aikins [6] has this to say about EMYCIN systems:

> 'One reason often cited for using production rules is that there is no direct interaction of one rule with the others, a characteristic which facilitates adding rules to the knowledge base or modifying existing rules. In practice, however, the rules are actually highly interconnected. If we want to add or modify a rule, we must first identify the set of rules that could invoke it and also the rules that it invokes in turn, and then determine whether changes in these rules also must be made. If we modify one or more of these rules, then the process repeats.'

RETRO possesses an explicit representation, with its knowledge base organised into groups of knowledge dealing with particular situations, so additional rules or new rule groupings may be added simply without reference to the rest of the knowledge base.

12.1.4 Explanation

Any expert system is unlikely to be accepted by the community if an adequate explanation of its performance is lacking. In RETRO the comprehensible control structure permits the explanations to be generated directly from the performance kmowledge.

12.2 Future work

Several ways have been mentioned in which RETRO may be developed. Other developments could be:

a. All rules used by RETRO have been derived manually, by studying a number of problems. Whether any rule induction procedure could be used is a matter for consideration.

b. It has been stated that the purpose of initial conditions is to impose constraints, without which it would not be possible to solve the given question. An intriguing variation would be to be given the question and the answer, then deduce what initial conditions are necessary for the answer to be true.

12.3 Summary

A program, RETRO, has been introduced that is capable of solving a number of problems in the domain of retrograde-analysis. The program is based on the concept of SEs, which direct attention to the questions that must be asked in order to effect a solution, analysis of published problems indicating that the number of SEs in this domain is small.

The solution of retrograde chess problems lies in (a) recognising an SE, (b) knowing what questions are pertinent to that SE, and (c) realising that there may be interactions between SEs, for the answer to a question may change the board situation, indicating another SE.

The work here has been concentrated on deriving rules to recognise SEs, knowing what questions to ask for each SE, and how to answer them. The rules used to recognise SEs and the rules used by SEs are grouped according to function, thus facilitating easy amendment and deletion without the 'knock-on' effect prevalent in EMYCIN type systems.

In particular, everything written is about Version I of RETRO, not a theoretical Version 'N', which has not been implemented.

13.0 REFERENCES

The following abbreviations are used:

ESMA Michie, D. (ed.) (1979), Expert Systems in the Micro-electronic Age, Edinburgh University Press.

HPP (1980), Stanford Heuristic Programming Project.

1. Smullyan, R. (1980), The Chess Mysteries of Sherlock Holmes, Hutchinson.

2. Smullyan, R. (1983), The Chess Mysteries of the Arabian Knights, Hutchinson.

3. Dawson, T.R., and Hunsdorfer, W. (1915), Retrograde-Analysis: A study, Whitehead and Miller.

4. McDermott, D. (1976),'Artificial Intelligence meets Natural Stupidity', SIGART Newsletter, No. 57, April 1976.

5. Shortliffe, E.H. (1976), Computer-based Medical Consultations: MYCIN, New York, American Elsevier/North Holland.

6. Aikins, J.S. (1980), 'Prototypes and Production Rules: A knowledge representation for computer consultations', HPP Memo, HPP-80-17.

7. Sloman, A. (1979), 'Epistemology and Artificial Intelligence', ESMA, pp. 235-141.

8. Young, R. (1979), 'Production Systems for modelling Human Cognition', ESMA, pp. 35-45.

9. Davis, R. (1982), 'Expert Systems: Where are we? And where do we go from here?', MIT Artificial Intelligence Laboratory, AI Memo No. 665.

10. Filman, R. (1979), 'The Interaction of Observation and Inference', Ph.D. Thesis, Stanford University.

11. Van Melle, W. (1979), 'A Domain-independent Production Rule System for Consultation Programs', Proceedings of the Sixth International Joint Conference on Artificial Intelligence, Tokyo, Japan, 1979.

12. Feigenbaum, E.A. (1979), 'Themes and Case Studies of Knowledge Engineering', ESMA, pp. 3-25.

13. Kunz, J.C. et al. (1978), 'A Physiological Rule-based System for interpreting Pulmonary Function Test Rules', HPP Memo, HPP-78-19.

14. Van Melle, W. et al. (1981), The EMYCIN Manual, Department of Computer Science, Stanford University, STAN-CS-81-885.

15. Alvey, P. (1983), 'The Problems in designing a Medical Expert System', Expert Systems '83 - Proceedings of British Computer Society Expert Systems Group Conference.

16. Weyhrauch, R.W. (1977), A Users' Manual for FOL, Stanford Artificial Intelligence Laboratory, Memo 2351.

17. Marr, D. (1976), 'Artificial Intelligence - A personal view', MIT Artificial Intelligence Laboratory, AI Memo No. 35.

18. Bramer, M. (1984), 'A survey and critical review of expert systems research' in D. Michie (ed.), Introductory Readings in Expert Systems, pp. 3-29; London, Gordon and Breach.

19. Goldstein, I.P., and Roberts, R.B. (1977), 'NUDGE, A Knowledge-based Scheduling Program', Proceedings of the Fifth International Joint Conference on AI, Cambridge, Mass.

20. Lenat, D.B. (1976), 'AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search', STAN-CS-76-570 (AI Memo No. 256), Stanford University, July.

21. Winston, P.H. (1977), Artificial Intelligence, Addison-Wesley.

22. Minsky, M. (1975), 'A framework for representing knowledge' in P. Winston (ed.), The Psychology of Computer Vision, New York, McGraw Hill.

23. Polya, G. (1973), How to solve it, Princeton University Press.

24. Bundy, A. (1983), The Automation of Mathematical Reasoning, Academic Press.

APPENDIX A

PAWN PROMOTION RULES

Rule 1 If the pawns (of one colour) have captured all opposition pieces that can be captured on board
and these pieces include a pawn that could not have reached a capture square
then an opposition pawn promoted.

Rule 2 If the pawns (of one colour) have captured 8 pieces
and an opposition pawn could not have reached a capture square
then an opposition pawn promoted.

Rule 3 If there are more than 2 rooks or more than 2 bishops or more than 2 knights or more than 1 queen (of one colour) on board
then a pawn promoted.

Rule 4 If one side has 2 bishops or more travelling on the same colour square
then a pawn promoted.

Rule 5 If one side has a bishop captured on its home square
and there is a bishop on board travelling on the same colour
then a pawn promoted.

Rule 6 If all pawn captures are on one colour
and these captures include an opposition pawn that could not have reached a capture square
then a pawn promoted.

Rule 7 If a pawn captured on promoting
and the only piece it could have captured is an opposition pawn
then an opposition pawn promoted.

Rule 8 If a pawn made N captures from leaving its home square to capturing on promoting
and the only pieces available to be captured were N opposition pawns
then an opposition pawn promoted.

APPENDIX B

RULES TO DETERMINE THE LEGALITY OF REVERSE MOVES

Rule 9 If a piece is given in the initial conditions as not moving
then conclude that all moves for this piece are illegal.

Rule 10 If a king is adjacent to king
then conclude that the move is illegal.

Rule 11 If the opposing king is in check
then conclude that the move is illegal.

(A piece will not move from a checking position.)

Rule 12 If own king is in check and no way found for this to happen
then conclude that the move is illegal.

Rule 13 If, after the reverse move, there are too many pawn captures
then conclude that the move is illegal.

Rule 14 If the number of pieces captured on board is less than
the number of opposition pawn captures
then conclude that the move is illegal.

Rule 15 If no captures are permitted
then conclude that all reverse move pawn captures are illegal.

Rule 16 If a pawn reverse moves to a home square
and constrains a piece that was captured on board
then conclude that the move is illegal.

APPENDIX C

'CASTLE' RULES

Rule 17 If a castle flag is set
and the last move is known
then delete all reverse moves for the possible castle pieces.

Rule 18 If a pawn has promoted on a square that contains a possible castle piece
then this piece must have moved.

Rule 19 If the promoted piece is a rook, constrained to its home square, and the king moved to let it out
then a castle is not possible.

Rule 20 If the promoted piece is a rook
and this rook is on a home square in a castle position
then a castle is not possible.

Rule 21 If a king moved to let out a rook that was captured on board
then a castle is not possible.

Rule 22 If a pawn promoted
and crossed the last but one rank such that the opposition king must have been in check
then a castle is not possible.

Rule 23 If an original rook has been captured on its home square
then a castle is not possible.

APPENDIX D

RULES TO DETERMINE IF A PROMOTED PIECE IS ON BOARD

Rule 24 If a pawn captured on its first move and could only have captured one opposing pawn
and this opposing pawn could not have made any captures
and the home square of the capturing pawn is on the same file as opposing pawn promotion
then the pawn captured before the opposing pawn promoted.

Rule 25 If a pawn captured on its first move
and could only have captured one opposing pawn
and the opposing pawn made N captures in promoting, where N is the number of pieces constrained by the capturing pawn on its home square
then the pawn captured before the opposing pawn promoted.

APPENDIX E

RULES TO DETERMINE THE PROMOTED PIECE

Rule 26 If there are 3 rooks or 3 bishops or 3 knights or 2 queens on board
then one of these must be promoted.

Rule 27 If there is more than 1 bishop on the same colour square
then one must be promoted.

Rule 28 If there is a bishop on board with its home square constrained by pawns
then it must be a promoted bishop.

Rule 29 If a piece on board is being considered as a possible promotion piece
then it must have been able to move from the promotion square.

APPENDIX F

SOME OF THE MORE IMPORTANT FUNCTIONS USED BY RETRO

BAMK — Given that a king is in check. If the checking piece is moved away from the checking line and the same king is still in check, then a capture is implied. Returns true or false.

CAPSQ — On what square did any pawn captures occur?
Returns lists for Black and White, which may be null;
e.g. [[WP h2][g3][f2]]].

The White pawn from h2 made captures on g3 and f2.

CPGTOB — Returns lists of pieces that were captured on board.

GENCON — Returns lists of constrained pieces together with squares to which constrained:

a. constrained rooks;
b. rooks not constrained if other pieces can move;
c. constrained bishops;
d. constrained queen.

HOMESQ — To find the home square of a given piece on xN;
e.g. If WQ then d1

```
elseif WB then check colour of square xN against c1 and f1
elseif WR then use GENCON (above)
          if [WR a1] constrained then h1
          elseif[WR h1] constrained then a1
else nil.
```

ISKCHK — Is a king in check?
If king in check then returns, for example [WK BB b1]
(the White king is in check from the Black bishop on b1)
or if a king is not in check then null
or if 2 kings are in check (illegal position) then false.

KCHECK — A suite of programs to determine how a king may have been placed in check:

CANCHK — Can the checking piece be moved to the check position other than along the checking line?

MOVCHK — Could a non-checking piece have moved from the checking line to discover check then be captured by the king?

MOVLIN — Could a non-checking piece have moved from the checking line to discover check?

PROM — Could a pawn have moved from the checking line to discover check, then promoted?

MINPWN For Black and White returns lists [X Y Z]
where X is the number of pawn captures
Y is the number of captures on White squares
Z is the number of captures on Black squares.
Also lists of home squares for pawns unaccounted for.

PCMOV Reverse move generator - generates all possible reverse moves irrespective of legality.

PROMCP If a pawn captured on promoting,can it be deduced that for each piece captured an opposition pawn promoted?

SCANTB Scan initial and final flags and printing the result of deduction.
e.g. For a 'castle' problem with two possible castles, BCFLGX = 2, BCFLG = 2. After applying 'castle' rules BCFLG will be reduced if any rules have been triggered.
If BCFLGX > BCFLG and BCFLG = Ø then Black cannot castle;
or if BCFLGX/ = Ø and not (BCAS null) then castle possible with rook in BCAS.

WCAPPN What was captured by pawn PWN on square (x, y)?
Allowance is made for possible pawn promotions, and,for instance, pieces that place the opposition king in check and no way found for this to happen.

WHSQPP On what square did a pawn promote?
Returns lists of:

a. possible promotion squares;
b. possible capture squares (if captured on promoting);
c. squares on which the pawn may have crossed the last but one rank.

The following functions deal exclusively with bishops:

BICOL If 2 bishops captured on board returns [2]
elseif Ø bishops captured on board returns [ ]
elsereturns (X, Y) of bishop on board or (X, Y) of bishop captured at home.

BCHSQ Returns lists of bishops captured at home.

DBISH Could a bishop have been captured by a pawn on promoting?

LCONS Return list of squares of constrained bishops.

APPENDIX G

THE WORD PARSERS PARS AND PARSEC

The word parser PARS is simple, but it works. PARS comprises the following suite of programs:

```
FUNCTION PARS XL;
Vars N;
   If MEMB ('LAST', XL) and MEMB ('MOVE', XL) then FNINT (XL) → N;
      If N.ISINTEGER then N → NC;  N → CP;  close;
   elseif MEMB ('WHAT', XL) and MEMB ('MISSING', XL) then FNLST (XL);
      1 → MSFLG;  1 → MSFLGX;
   elseif MEMB ('WHAT', XL) and MEMB ('SQUARE', XL) and MEMB
      ('CAPTURED", XL) then FNLST (XL);  MISPC → CAPPC;  Ø → MISPC;
   elseif MEMB ('LAST', XL) and MEMB ('MOVES', XL) then FNINT (XL) → N;
      If N.ISINTEGER then N → NC;  N → CP;  close;
   elseif MEMB ('CASTLE', XL) or MEMB ('CASTLING', XL) then FNCAS (XL) → N;
      If N = 'B' then lengt (BCAS) → BCFLG;  copyall (BCFLG) → BCFLGX;
      elseif N = 'W' then lengt (WCAS) → WCFLG;  copyall (WCFLG) → WCFLGX;
      else lengt (BCAS) → BCFLG;  lengt (WCAS) → WCFLG;
           copyall (BCFLG) → BCFLGX;  copyall (WCFLG) → WCFLGX;
      close;
   elseif TWO (XL) then LOCAT (XL);
   elseif MEMB ('PROMOTED', XL) and MEMB ('BOARD', XL) then 1 → PPONB;
           copyall (PPONB) → PPONBX;
        else   .NONQ;
        close;
end;

FUNCTION MEMB X XL;
COMMENT  If x is a member of XL then true, else false;
         If XL.null then false;
         elseif X = XL.HD then true;
         else MEMB (X, TL (XL));
         close;

end;

FUNCTION FNCAS XL;
COMMENT  possible castling problem - is it black or white or both;
         If XL.null then 'U';  return;
         elseif XL.HD = 'B' or XL.HD = 'BLACK' then 'B';
         elseif XL.HD = 'W' or XL.HD = 'WHITE' then 'W';
         else FNCAS (TL (XL));
         close;

end;

FUNCTION FNINT XL;
COMMENT  Look through XL to see if contains integer;
         If XL.null then 'A';
         elseif XL.HD.ISINTEGER then XL.HD;
         else FNINT (TL (XL));
         close;
end;
```

```
FUNCTION FNLST XL;
COMMENT  missing piece - find if square given
         if square place is MISPQ else MISPQ null
         if piece place in MISPC else MISPC zero;

Vars YL;  XL → YL;

Loop:    If YL.null then return;
         elseif YL.HD.ISLIST then YL.HD → MISPQ;
         elseif BRW (YL.HD) then YL.HD → MISPC;
         else TL (YL) → YL;  goto loop;
         close;

end;

FUNCTION BRW X;
COMMENT  Find if piece given;
         If X = 'BP' or X = 'BB' or X = 'BN' or X = 'BR' or X = 'BK'
         or X = 'BQ' or X = 'WP' or X = 'WB' or X = 'WN' or X = 'WR'
         or X = 'WK' or X = 'WQ'
         then true;
         else false;
         close;

end;

FUNCTION TWO XL;
COMMENT  if there are two lists in XL then possible location problem;

Vars YL YLH;  nil → PLOC;  copyall (XL) → YL;
loop:    If YL.null then goto L1;
         else dest (YL) → YL → YLH;
              If YLH.ISLIST then YLH::PLOC → PLOC;
              close;
         goto loop;
         close;

L1:      If lengt (PLOC) = 2 then true;
         else nil → PLOC;  false;
         close;

end;

FUNCTION LOCAT XL;
COMMENT  location problem determine piece;

Vars  YL XLH H1 H2;  nil → YL;
loop:    If XL.null then false;  return;
         else dest (XL) → XL → XLH;
              If BRW (XLH) then PLOC.HD → H1;  PLOC.TL.HD → H2;
              XLH::H1 → H1;  XLH::H2 → H2;  H2::YL → YL;  H1::YL → YL;
              YL → PLOC;  true;  1 → LOCFLG;  6 → CP;  return;
              close;
         goto loop;
         close;

end;
```

```
FUNCTION NONQ;
n1(1);  prstring ('Unable to parse question, but deductions will be
carried out!');  n1(1);
end;
```

```
FUNCTION PARSEC XL;
Comment  To parse initial conditions;

If MEMB ('LAST', XL) and MEMB ('MOVE', XL) then LMV (XL);
elseif MEMB ('LAST', XL) and MEMB ('MOVED', XL) then LMV (XL);
elseif MEMB ('CAPTURE', XL) or MEMB ('CAPTURES', XL) then CAPM (XL);
elseif MEMB ('CAN', XL) and MEMB ('CASTLE', XL) then BWCANC (XL);
elseif MEMB ('COLOUR', XL) and MEMB ('MOVED', XL) then COLM (XL);
elseif MEMB ('COLOUR', XL) and MEMB ('MOVE', XL) then COLM (XL);
elseif MEMB ('ODDS', XL) then ODDM (XL);
elseif MEMB ('ORIGINAL', XL) then ORPM (XL);
elseif MEMB ('CHECK', XL) then CHM (XL);
elseif MEMB ('PROMOTED', XL) then 0 → PPOB;
elseif MEMB ('MOVED', XL) and MEMB ('NOT', XL) then MVL (XL);
elseif MEMB ('UNDERPROMOTIONS', XL) then 0 → UPFLG;
else NONIC;
close;
end;
```

The functions called perform the following operations:

NONIC Prints 'unable to parse initial conditions'

MVL XL contains pieces that have not moved, locate and place in PNM

BWCANC Given that black or white can castle. Add to pieces not moved list PNM

ORPM Given that some pieces on board are original. Add to ORP

ODDM Odds were given. Add to ODG

COLM Piece(s) not moved off own colour. Add to COLLST

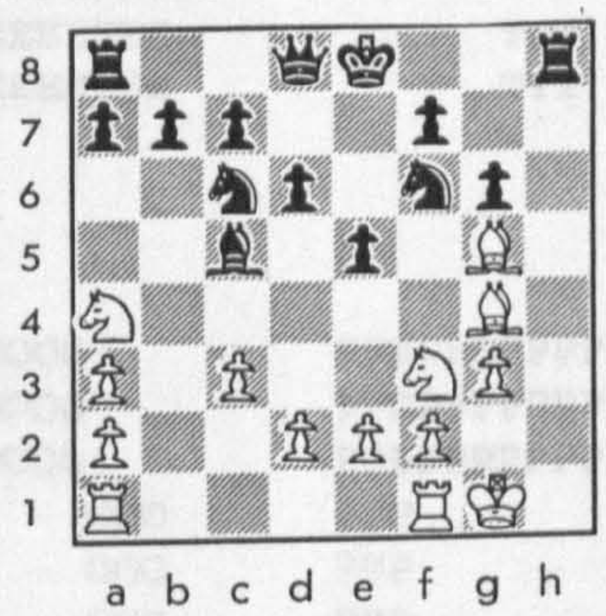
LMV Who moved last. Place in LM

CAPM Captures permitted?

APPENDIX H

THE FINAL BOW

Mr. Sherlock Holmes, being a keen solver of retrograde-analysis problems, graciously allowed himself to be quoted throughout the course of this work. What is not so well known is that his arch enemy, Professor Moriarty, is also an expert solver, and is reputed to have solved the following extremely complex problem in 3-4 minutes:



Question: Is there a promoted piece on board?

Initial conditions: White has just castled.

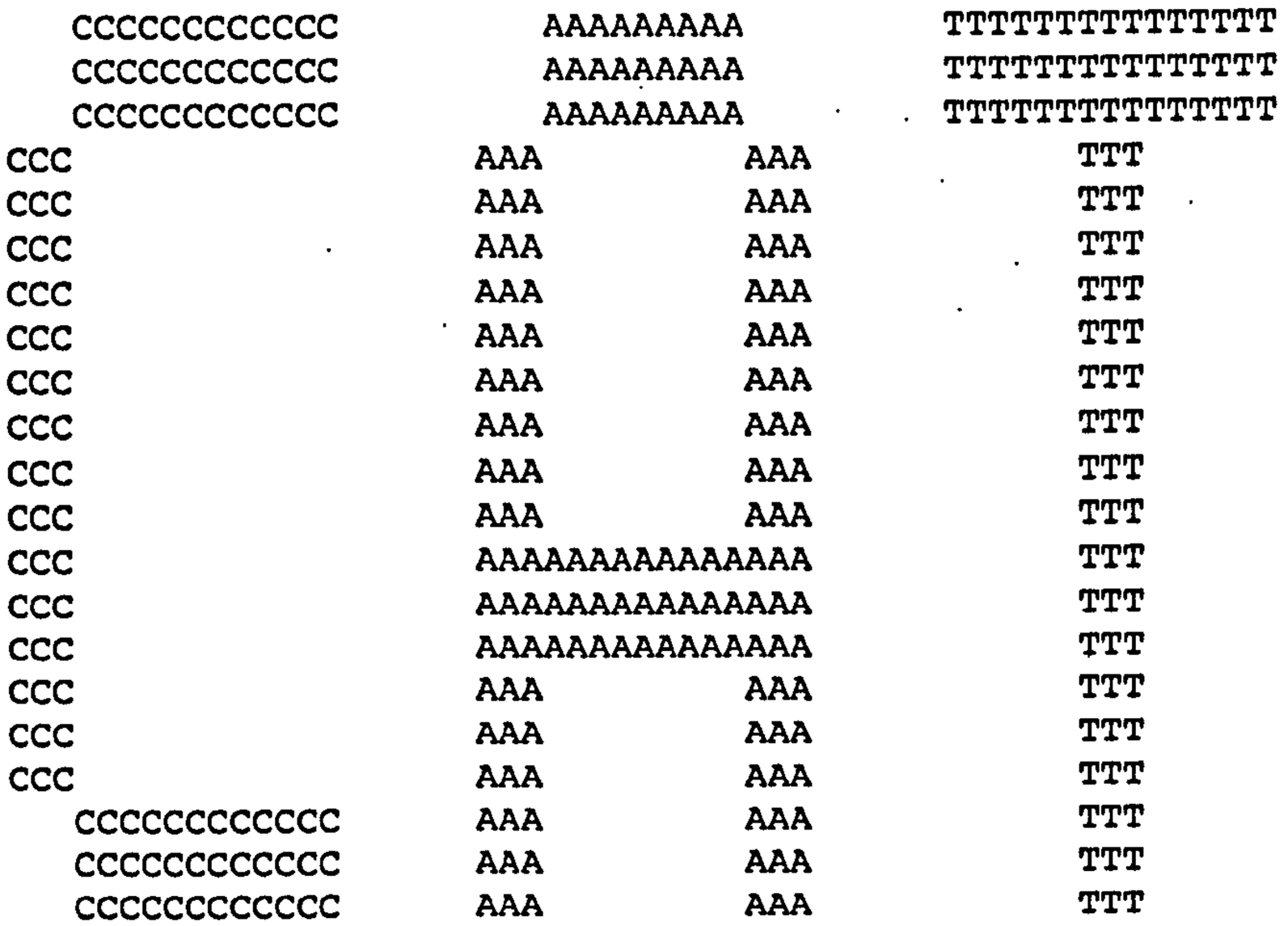
The reader may care to consider the problem. (RETRO, alas, is not yet able to solve it.)

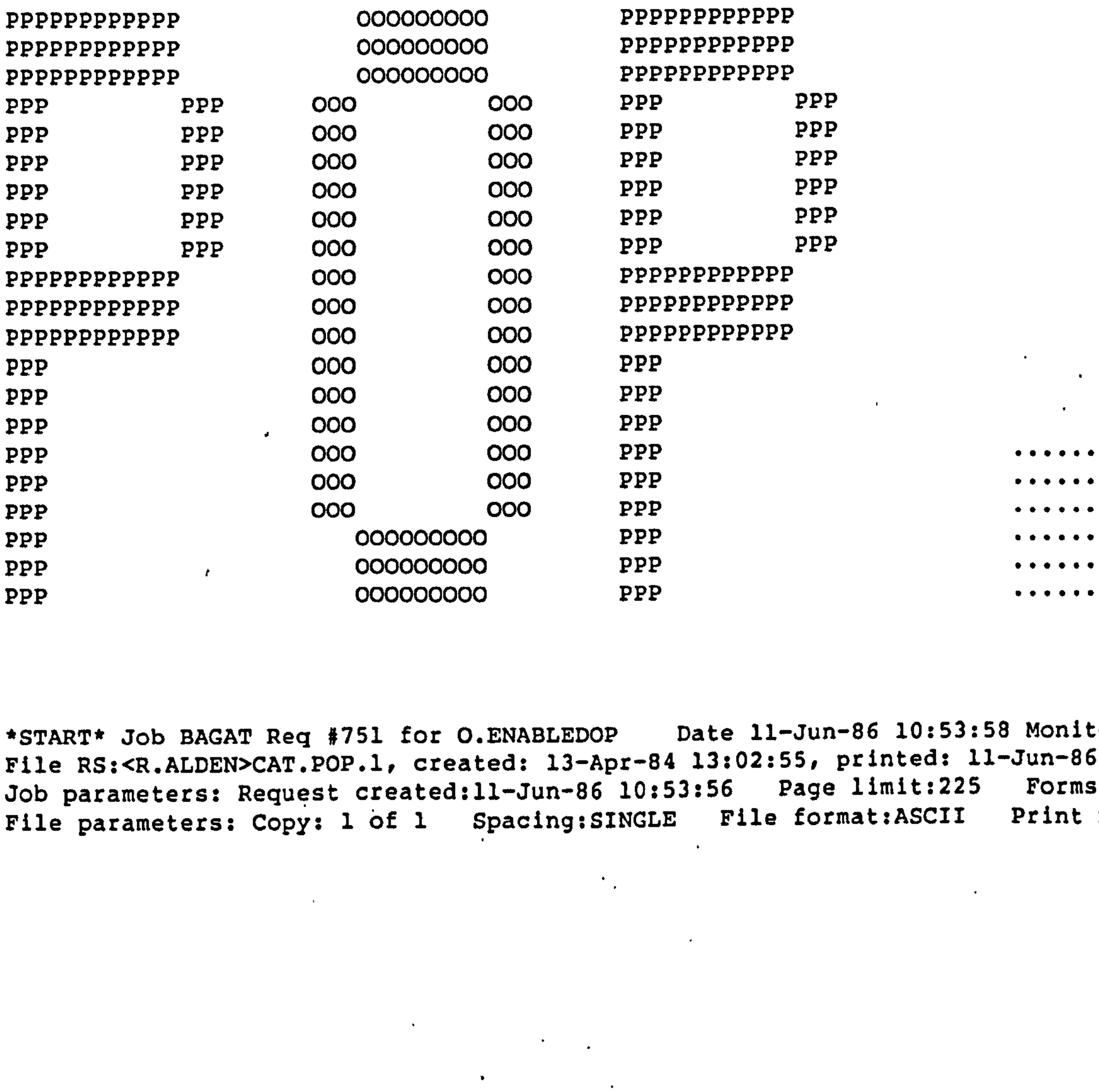
```
   SSSSSSSSSSS       EEEEEEEEEEEEEEEE     TTTTTTTTTTTTTTTT      UUU         UUU
   SSSSSSSSSSSS      EEEEEEEEEEEEEEEE     TTTTTTTTTTTTTTTT      UUU         UUU
   SSSSSSSSSSSS      EEEEEEEEEEEEEEEE     TTTTTTTTTTTTTTTT      UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
SSS                  EEE                        TTT             UUU         UUU
   SSSSSSSS          EEEEEEEEEEEE               TTT             UUU         UUU
   SSSSSSSS          EEEEEEEEEEEE               TTT             UUU         UUU
   SSSSSSSS          EEEEEEEEEEEE               TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
            SSS      EEE                        TTT             UUU         UUU
SSSSSSSSSSS          EEEEEEEEEEEEEEEE           TTT             UUUUUUUUUUUUUUU
SSSSSSSSSSS          EEEEEEEEEEEEEEEE           TTT             UUUUUUUUUUUUUUU
SSSSSSSSSSS          EEEEEEEEEEEEEEEE           TTT             UUUUUUUUUUUUUUU

PPPPPPPPPPPP            OOOOOOOOO            PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO            PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO            PPPPPPPPPPPP
PPP         PPP      OOO         OOO         PPP         PPP
PPP         PPP      OOO         OOO         PPP         PPP
PPP         PPP      OOO         OOO         PPP         PPP
PPP         PPP      OOO         OOO         PPP         PPP
PPP         PPP      OOO         OOO         PPP         PPP
PPPPPPPPPPPP         OOO         OOO         PPPPPPPPPPPP
PPPPPPPPPPPP         OOO         OOO         PPPPPPPPPPPP
PPPPPPPPPPPP         OOO         OOO         PPPPPPPPPPPP
PPP                  OOO         OOO         PPP
PPP                  OOO         OOO         PPP
PPP                  OOO         OOO         PPP
PPP                  OOO         OOO         PPP                      ......
PPP                  OOO         OOO         PPP                      ......
PPP                  OOO         OOO         PPP                      ......
PPP                     OOOOOOOOO            PPP                      ......
PPP                     OOOOOOOOO            PPP                      ......
PPP                     OOOOOOOOO            PPP                      ......

*START* Job LISTS1 Req #752 for O.ENABLEDOP    Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>SETUP.POP.1, created: 28-Mar-83 13:20:37, printed: 11-Jun-86 11
Job parameters: Request created:11-Jun-86 10:53:57   Page limit:111   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
ALLRULES          [RULE001  RULE002  RULE003  RULE004  RULE005   RULE006
RULE007  RULE008  RULE009  RULE010  RULE011  RULE012  RULE013  RULE014
RULE015  RULE016  RULE017  RULE018  RULE019  RULE020  RULE021  RULE022
RULE023 RULE024 RULE025 RULE026 RULE027 RULE028 RULE029]
```

```
RULE001   category  PROMRULE
RULE001   premise   [RULE1]
RULE001   explain   [R1EX]

RULE002   category  PROMRULE
RULE002   premise   [RULE2]
RULE002   explain   [R2EX]

RULE003   category  PROMRULE
RULE003   premise   [RULE3]
RULE003   explain   [R3EX]

RULE004   category  PROMRULE
RULE004   premise   [RULE4]
RULE004   explain   [R4EX]

RULE005   category  PROMRULE
RULE005   premise   [RULE5]
RULE005   explain   [R5EX]

RULE006   category  PROMRULE
RULE006   premise   [RULE6]
RULE006   explain   [R6EX]

RULE007   category  PROMRULE
RULE007   premise   [RULE7]
RULE007   explain   [R7EX]

RULE008   category  PROMRULE
RULE008   premise   [RULE8]
RULE008   explain   [R8EX]

RULE009   category  LEGRULE
RULE009   premise   [RULE9]
RULE009   explain   [R9EX]

RULE010   category  LEGRULE
RULE010   premise   [RULE10]
RULE010   explain   [R10EX]

RULE011   category  LEGRULE
RULE011   premise   [RULE11]
RULE011   explain   [R11EX]

RULE012   category  LEGRULE
RULE012   premise   [RULE12]
RULE012   explain   [R12EX]

RULE013   category  LEGRULE
RULE013   premise   [RULE13]
RULE013   explain   [R13EX]

RULE014   category  LEGRULE
RULE014   premise   [RULE14]
RULE014   explain   [R14EX]

RULE015   category  LEGRULE
RULE015   premise   [RULE15]
RULE015   explain   [R15EX]

RULE016   category  LEGRULE
RULE016   premise   [RULE16]
RULE016   explain   [R16EX]
```

```
RULE017  category  CASRULE
RULE017  premise   [RULE17]
RULE017  explain   [R17EX]

RULE018  category  CASRULE
RULE018  premise   [RULE18]
RULE018  explain   [R18EX]

RULE019  category  CASRULE
RULE019  premise   [RULE19]
RULE019  explain   [R19EX]

RULE020  category  CASRULE
RULE020  premise   [RULE20]
RULE020  explain   [R20EX]

RULE021  category  CASRULE
RULE021  premise   [RULE21]
RULE021  explain   [R21EX]

RULE022  category  CASRULE
RULE022  premise   [RULE22]
RULE022  explain   [R22EX]

RULE023  category  CASRULE
RULE023  premise   [RULE23]
RULE023  explain   [R23EX]

RULE024  category  IBPRUL
RULE024  premise   [RULE24]
RULE024  explain   [R24EX]

RULE025  category  IPBRUL
RULE025  premise   [RULE25]
RULE025  explain   [R25EX]

RULE026  category  WPPRUL
RULE026  premise   [RULE26]
RULE026  explain   [R26EX]

RULE027  category  WPPRUL
RULE027  premise   [RULE27]
RULE027  explain   [R27EX]

RULE028  category  WPPRUL
RULE028  premise   [RULE28]
RULE028  explain   [R28EX]

RULE029  category  WPPRUL
RULE029  premise   [RULE29]
RULE029  explain   [R29EX]
```





```
FUNCTION PRINTLS XL;
        vars HL;REV(XL)->XL;
loop:   If XL.null then return;
        else XL.HD->HL;nl(1);pr(HL.HD);sp(1);pr("from");sp(1);
        pr("(");pr(HL.TL.TL.TL.HD);pr(",");pr(HL.TL.TL.TL.TL.HD);pr(")");
        sp(1);pr("to");sp(1);pr("(");pr(HL.TL.HD);pr(",");pr(HL.TL.TL.HD);
        pr(")");nl(1);
        close;
        TL(XL)->XL;goto loop;
end;


FUNCTION CHKFLG;
COMMENT looks through flags to see what must be deduced
                if all flags zero (i.e. nothing more to deduce),returns false;
        If BCFLG/=0 or WCFLG/=0 then .CASTLE;close;
        If MSFLG=1 then .LKFMP;close;
        If PPONB/=0 then .ISPPB;close;

        If ILLFLG=1 and LOCFLG=1 and TRANS=0 then copyall(PLOCR)->PLOC;
        1->TRANS;copyall(AGC)->AGX;false;.RFLGTAB;return;
        elseif LOCFLG=1 and TRANS=1 and AGC=AGX then false ;return;
        elseif ILLFLG=1 then false;.RFLGTAB;return;
        elseif BCFLG=0 and WCFLG=0 and MSFLG=0 and PPONB=0 and LOCFLG=0
                and CAPPC=0 then false;
        else true;
        close;

        end;

FUNCTION LKFMP;
COMMENT look for missing piece
                first check colour
                then try for piece;
        .COLMP;.LFMP;

COMMENT if piece found MSFLG set to false;

end;


FUNCTION RFLGTAB;
COMMENT restore original flag table to consider other moves;

        COPYALL(BCFLGX)->BCFLG;
        COPYALL(WCFLGX)->WCFLG;
        COPYALL(PPONBX)->PPONB;
end;
FUNCTION CATPRB;

        .RFLGTAB;
COMMENT  categorise problem.if last move known
                and a cstle situation apply castle rules;

        If not(.CHKFLG) and LOCFLG=0 then exit;
VARS WC BC ANS ;.SETP;LENGT(BLEFT)->BC;LENGT(WLEFT)->WC;


        If .CHCAT1 then .CAT1;
        else .DPPROM->PRLST;
                If not(PRLST.null) then .CAT2;
                else ISKCHK(BLIST,WLIST)->CHI->KC;
                        If not(CHI=false) and not(CHI.null) then CAT3(CHI,KC);
```

```
                        else .CAT4;
                        close;
                close;
        close;

END;
FUNCTION CHCAT1;
COMMENT to check SE1 is it possible to determine what pieces the
                pawns have captured.
                assumes all pawns on board.
                if (e.g.) n bp captures then n pieces available for capture
                on board (deleting doubles from list);

        If NPB/=8 or NPW/=8 then false;
        elseif WPTT=BTOT and BPTT=WTOT then true;
        elseif CHKDB(WPTT,BLEFT) and CHKDB(BPTT,WLEFT) then true;
        else false;
        close;
END;

FUNCTION CHKDB PCP XL;

        vars YL;nil->YL;

loop:   If not(XL.null) then XL.HD::YL->YL;DELT(XL.HD,XL)->XL;
        goto loop;
        elseif PCP=lengt(YL) then true;
        else false;
        close;

end;

FUNCTION HOMESQ PC HL;
COMMENT is it possible to determine the home square of piece pc
                used in catla;

        vars P QC BC RU RC;
        BORW(PC)->P;

        If P="B" then goto L1;
        elseif PC="WQ" then [4 1];
        elseif PC="WB" and CLSQT(HL.TL.HD,HL.TL.TL.HD,3,1) then [3 1];
        elseif PC="WB" then [6 1];
        elseif PC="WR" then GENCON("W")->QC->BC->RU->RC;
              If RC.null then nil;
              elseif equal([WR 1 1],RC.HD.HD) then [8 1];
                      [WR 8 1]::RKLIST->RKLIST;
              else [1 1];[WR 1 1]::RKLIST->RKLIST;
              close;
        else nil;
        close;return;


L1:     If PC="BQ" then [4 ];
        elseif PC="BB" and CLSQT(HL.TL.HD,HL.TL.TL.HD,3,8) then [3 8];
        elseif PC="BB" then [6 8];
        elseif PC="BR" then GENCON("B")->QC->BC->RU->RC;
                If RC.null then nil;
                elseif equal([BR 1 8],RC.HD.HD) then [8 8];
                [BR 8 8]::RKLIST->RKLIST;
                else [1 8];[BR 1 8]::RKLIST->RKLIST;
                close;
        else nil;
```

```
        close;
end;

FUNCTION SETBC XL;
VARS HL HHL;
IF XL.NULL THEN NIL;
ELSE XL.HD->HL;HL.TL->HHL;
[%HL.HD.HD,HHL.HD.HD,HHL.HD.TL.HD,HL.HD.TL.HD,HL.HD.TL.TL.HD%]::SETBC(TL(XL));
CLOSE;
END;
FUNCTION UPDATEX PL XL YL=>YL;
COMMENT check before performing reverse move if there is a
                piece on square,if so,swop;

        vars    X Y;
        PL.TL.TL.TL.HD->X;PL.TL.TL.TL.TL.HD->Y;
        If BOARD(X,Y)/="BLANK" then [%BOARD(X,Y),X,Y%]::YL->YL;
                SWOP(X,Y,PL.TL.HD,PL.TL.TL.HD);
        else UPDATE(PL,XL);
        close;
END;
FUNCTION RESTORX HL YL;
VARS X Y ANS;
HL.TL.TL.TL.HD->X;HL.TL.TL.TL.TL.HD->Y;CKMTAB(X,Y,YL)->ANS;
IF NOT(ANS.NULL) THEN SWOP(X,Y,HL.TL.HD,HL.TL.TL.HD);ELSE RESTOR(HL);CLOSE;
END;
FUNCTION SWOP X1 Y1 X2 Y2;
VARS P XXL TMP1 TMP2;
BORW(BOARD(X,Y))->P;
IF P="B" THEN BLIST->XXL;ELSE WLIST->XXL;CLOSE;
BOARD(X1,Y1)->TMP1;BOARD(X2,Y2)->TMP2;TMP1->BOARD(X2,Y2);
TMP2->BOARD(X1,Y1);MXYLST(TMP1,X1,Y1,X2,Y2,XXL);MXYLST(TMP2,X2,Y2,X1,Y1,XXL);
END;
FUNCTION MXYLST PC XX1 XX2 XX3 XX4 XXL;
IF XXL.HD.HD=PC AND XXL.HD.TL.HD=XX1 AND XXL.HD.TL.TL.HD=XX2
THEN XX3->XXL.HD.TL.HD;XX4->XXL.HD.TL.TL.HD;
ELSE MXYLST(PC,XX1,XX2,XX3,XX4,TL(XXL));CLOSE;
END;
FUNCTION CKMTAB X Y YL;
IF YL.NULL THEN NIL;
ELSEIF YL.HD.TL.HD=X AND YL.HD.TL.TL.HD=Y THEN YL.HD;
ELSE CKMTAB(X,Y,TL(YL));CLOSE;
END;
FUNCTION PWNCPM PN LIS PSQ PPL LFC XHH;
VARS XS XP XL XPC YL HHSQ XXL;[%PPL.HD.TL.HD,PPL.HD.TL.TL.HD%]->HHSQ;
LIS.HD.HD->XS;PSQ.HD->XP;
IF PN="WP" THEN BTOT-WNUM.HD->NCAP;BLEFT->XL;DELT("BP",XL)->YL;
ELSE WTOT-BNUM.HD->NCAP;WLEFT->XL;DELT("WP",XL)->YL;CLOSE;
PCAPRM(PN,LIS,PSQ,PPL,LFC)->XPC;
IF XS=XP AND XS=XHH AND NCAP=0 THEN PWNP1(PN);
ELSEIF XS=XP AND XS=XHH AND NOT(NCAP<2) THEN WONWY(LIS,PN)->XXL;PWNP2(PN,XXL);
ELSEIF XS=XP AND XS=XHH AND NCAP<2 THEN PWNP1(PN);
ELSEIF XS=XP AND XS/=XHH THEN WONWY(LIS,PN)->XXL; PWNP2(PN,XXL);
ELSEIF XS/=XP AND XS=XHH AND NOT(NCAP>2) THEN PWNP5(PN,XPC);
ELSEIF XS/=XP AND EQUAL(XPC,YL) THEN WONWY(LIS,PN)->XXL;PWNP4(PN,XXL);
ELSEIF XS/=XP AND NOT(EQUAL(XPC,YL)) THEN WONWY(LIS,PN)->XXL;PWNP3(PN,XXL,XPC);CLOSE;
END;
FUNCTION CHCAT3;
ISKCHK(BLIST,WLIST)->CH->KI;
IF CH=FALSE OR CH.NULL THEN FALSE;ELSE TRUE;CLOSE;
END;
FUNCTION CAT4;
VARS ML CL;
IF LM="U" OR LENGT(MLIST)>1 THEN GOTO PRIN;
```

```
ELSEIF LM="W" THEN PCMOV(WLIST)->ML;ELSE PCMOV(BLIST)->ML;CLOSE;
IF NOT(ML.NULL) THEN GOTO PRIN;CLOSE;
NL(2);PRSTRING('NO FURTHER REVERSE MOVE FOR!);SP(1);PR(LM);SP(1);
PRSTRING('SO!);SP(1);PR(LM);SP(1);PRSTRING('PIECE CAPTURED!);NL(1);
.WPDCAP->CL;
PRSTRING('THE ONLY PIECES THAT COULD CAPTURE THE !);SP(1);PR(LM);SP(1);
PRSTRING('PIECE ARE:-!);NL(1);CL=>;LCAST(CL);RETURN;
PRIN:   NL(2);PRSTRING('NO SES FOUND!);
END;
FUNCTION FCAP P;
IF P="W" THEN FORALL I 1 1 8 IF BOARD(1,7)="BP" AND BOARD(1,6)="BP"
THEN [%1,6%];CLOSE;CLOSE;
ELSE FORALL I 1 1 8 IF BOARD(I,2)="WP" AND BOARD(I,3)="WP" THEN [%I,2%];
CLOSE;CLOSE;CLOSE;
END;
FUNCTION LCAT2 PWN=>ANS;
VARS PC X CORD TC N CH;BORW(PWN)->PC;
IF PC="W" THEN WHTAB.HD->X;LENGTH(BLEFT)->N;WTOT.HD->TC;
ELSE BHTAB.HD->X;LENGTH(WLEFT)->N;BTOT.HD->TC;CLOSE;
FCAP(PC)->CORD;ABBS(CORD.HD-X)->CH;
IF CH>4 OR TC+CH>N THEN FALSE->ANS;ELSE TRUE->ANS;CLOSE;
END;
FUNCTION RVP HL;
VARS PC R1 R2 YL;BORW(HL.HD)->PC;
[%HL.HD,HL.TL.TL.TL.HD,HL.TL.TL.TL.TL.HD,HL.TL.HD,HL.TL.TL.HD%]->YL;
IF PC="B" THEN UPDATE(YL,BLIST);ELSE UPDATE(YL,WLIST);CLOSE;
ISKCHK(BLIST,WLIST)->R1->R2;
IF R2=KI THEN TRUE;ELSE FALSE;CLOSE;RESTOR(YL);
END;
FUNCTION RETRO;
0->POPCOMMENT;
COMMENT mainline program;
VARS BOARD NPB NPW WHCAP BLCAP WLEFT BLEFT LM CAP CP PLOC SFLAG BWL COLLST CS;
VARS MLIST CLIST NML NC CHI KC CHLIN KL WHTAB BHTAB BTOT WTOT PNM BLIST WLIST BC;
VARS WCAS BCAS PRLST NCAP HP MML MVL ODG ORP MLT IFLG PSQQ HSQ PPNFLG;0->PPNFLG;
VARS BDC WDC CPB CPW CPBX CPWX XYLIFS WMTAB BMTAB BHTAB1 WHTAB1 MISPQ BPPC ;
VARS WPPC UTILITY MISPC WPTT BPTT XCAPW XCAPB PROMXL WCFLG BCFLG ITM ILST ZL;
VARS    WHPFLG ILLFLG PLOCHD LOCFLG UPFLG PSQS PPIECE PPAWN PPOB BCFLGX WCFLGX;
vars    CAPBRD PPONB PPONBX ISPFLG LIFC R32FLG OMIT33 PPCAP XYLIFX WHSQFL;
vars    PHSQ MSPC AGC AGX TRANS PLOCR C3FLAG MSFLG MSFLGX MISCOL PPCOP CAPPC;
vars    RKLIST WPPC BPPC;
vars  WNUM BNUM WPT BPT BW NWLIST NBLIST; "N"->CS;
COMMENT set up initial variables;

        NIL->PNM;0->CP;"U"->LM;NIL->PLOC;NIL->COLLST;NIL->ORP;NIL->ODG;
        nil->XYLIFX;nil->MISCOL;0->TRANS;nil->PLOCR;0->AGX;0->AGC;nil->PHSQ;
        nil->PPCOP;
COMMENT DDB=dynamic data base.  LEGDEL=list of legality check
                deleted moves.LEGFLG=flag set in legality checker
                If =1 then place deleted moves in DDB
                else=0;

vars DDB LEGDEL LEGFLG DDBF ASKXL ;
COMMENT global variables are:-
        NPB/NPW no.of b/w pawns on board  WHCAP/BLCAP lists of pieces captured
        WLEFT/BLEFT lists of pieces captured on board
        LM if black moved last then "B", if white then "W" else "U"
        CAP     captures permitted "Y" or "N"
        CP      captures permitted for cp moves
        TRANS used for location probs.to indicate reverse list being used
        AGC     used in class.pop counts how many contexts perfirmed
        AGX     record of AGC when illegality occurs
        PLOCR   reverse PLOC list
        PLOC if location problem contains list of possible locations,else null
```

```
SFLAG used in functions UPDATE and RESTOR
COLLST  list of pieces not moving off own colour else null
MLIST   list of reverse moves
WHTAB/BHTAB     vacant pawn home squares
BTOT/WTOT       length of lists BLEFT/WLEFT
PNM     list of pieces given as not moving else null
BLIST/WLIST     list of pieces on board
BNUM/WNUM       lists of definite pawn captures
NWLIST/NBLIST   copies of BLIST/WLIST for restoring board
WCAS/BCAS       pieces in possible castle situationms
PRLST   print list of moves
NCAP    no of possible captures by promoting pawn
ODG     odds given
ORP     original pieces still on board
BDC/WDC lists of pawn capture squares
PPCOP if a pawn has captured in promoting,this list of poss. captures
MISCOL  missing piece,its colour
MISPQ/MISPC     missing piece square and missing piece(if known)
WPTT/BPTT       head of lists BNUM/WNUM
PROMXL  pawn promotion lists
ASKXL question posed    ITM     for itemread
ILST    list of initial conditions
PLOCHD  location problem - variable contains location being considered
UPFLG flag=0 when no underpromotions
LOCFLG  flag set if location problem
BCFLG/WCFLG flags set when castle problem
XYLIFS list or squares where promoted pawn crossed last rank but one.
PSQS    list of possible pawn promotion squares.
PPIECE  list of promoted pieces.
PPAWN   promoted pawn "WP"/"BP"
PPOB    if promoted piece on board 1,else 0
CAPBRD  set if a promoted piece was possible captured on board
PPONB   set if question "is a promoted piece on board"
ISPFLG  set to 1 when a promoted piece found to be on board,else 0
LIFC    list of squares on which a promoted pawn captured
R32FLG  set to 0 when rule 33 triggers else 0(see rule 34)
OMIT33 used in rule 33,omit rook on home square(constrained) - rule 32
PPCAP promoted pawn wp or bp or 0 - used by rule 7 - ppawn can
        be overwritten
MSPC on what square was piece MSPC captured
XYLIFX  copy of XYLIFS
WHSQFL flag for fn WHSQPP,if set not place result in ddb
PHSQ    home square of promoted pawn [x,y]
C3FLAG  used in  CAT3A to indicate that promotion square known
MSFLG/MSFLX     set when missing piece problem
CAPPC on what square was cappc captured
RKLIST  list of rooks captured on home square (catla)
WPPC/BPPC       count of pawns promoted
ILLFLG  flag set when illegal situation occurs.
        WHPFLG  flag set if a k is in check from pc and still in check if
                pc moves - implying a capture.set inBAMK;
NIL->DDB;0->ILLFLG;0->UPFLG;0->LOCFLG;NIL->LEGDEL;0->WHPFLG;0->LEGFLG;
NIL->PSQQ;nil->LIFC;NIL->XYLIFS;NIL->MISPQ;"Y"->CAP;NIL->WMTAB;NIL->BMTAB;NIL->HSQ;
        0->WCFLG;0->BCFLG;NIL->PSQS;NIL->PPIECE;0->PPAWN;1->PPOB;0->PPONB;
        0->CAPBRD;0->C3FLAG;0->MSFLG;0->MSFLGX;0->WHSQFL;0->PPCAP;0->PPONBX;
        nil->RKLIST;0->WPPC;0->BPPC;
.START;
COMMENT set up board;

LOOPY:NEWARRAY([1 8 1 8],LAMBDA,I,J;"BLANK";END;)->BOARD;
COMMENT get list of pieces on board;

nl(1);prstring('Enter black pieces!);nl(1);.listread->BLIST;
nl(2);prstring('Enter white pieces!);nl(1);.listread->WLIST;
```

```
COMMENT place pieces on board and display;
        .place2;.display;
COMMENT ask if board correct;
        nl(1);prstring('is the board correct ? <Y or N> :!);
        .itemread->BC;If BC="N" then goto LOOPY;close;
        .SETP;
COMMENT keep copy of original lists;
COPYALL(BLIST)->NBLIST;COPYALL(WLIST)->NWLIST;
COMMENT ask for the question;
        nl(1);prstring('what is the question!);nl(1);
        .listread->ASKXL;PARS(ASKXL);

COMMENT ask for initial conditions;
ICP:    nl(1);prstring('any initial conditions <Y or N> !);
IC:     .itemread->ITM;
        If ITM/="Y" and ITM/="N" then goto IC;
        elseif ITM="Y" then nl(1);prstring('please enter!);nl(1);
        .listread->ILST;PARSEC(ILST);
                nl(1);prstring('any more conditions <Y or N>!);
                nl(1);goto IC;
        close;

        1->ILLFLG;If LOCFLG=1 then REV(PLOC)->PLOCR;close;
LOC:    If not(PLOC.null) and ILLFLG=1 then PLOC.HD->PLOCHD;
                nil->DDB;nil->DDBFX;PLOCB(PLOC)->PLOC;
        elseif not(PLOC.null) and ILLFLG=0 and TRANS=0 then
                copyall(PLOCR)->PLOC;1->ILLFLG;goto LOC;
        elseif not(PLOC.null) and ILLFLG=0 and TRANS=1 then PRLOC(PLOCHD);.ASK;return;
        close;
        0->ILLFLG;

COMMENT If last move known - form reverse moves,else got to
                CATPRB to classify problem (module A);
        If LM="B" then PCMOV(BLIST)->MLIST;
        elseif LM="W" then PCMOV(WLIST)->MLIST;
        else .CATPRB;

COMMENT after classification & solution see if another location
                to be considered - if no,exit,else reset;

        If PLOC.null and LOCFLG=1 then PRLOC(PLOCHD);.ASK;return;
        elseif PLOC.NULL then .ASK;return;
        else COPYALL(NWLIST)->WLIST;COPYALL(NBLIST)->BLIST;.RSTB;
                0->PPAWN;nil->PPIECE;0->PPCAP;nil->PRLST;
                goto LOC;close;
close;

COMMENT if this is a castle problem delete any possible
                rook and king moves from subsequent analysis;
        If BCFLG/=0 or WCFLG/=0 then .CASTLE;close;

COMMENT perform legality check on reverse moves;
        [%[MLIST],MLIST,[R44EX]%]::DDB->DDB;1->LEGFLG;LEGCHK(CAP,MLIST)->MLIST;
        If MLIST.NULL then nl(1);prstring('No moves found!);nl(1);

        COPYALL(NWLIST)->WLIST;COPYALL(NBLIST)->BLIST;.RSTB;
        else COPYALL(MLIST)->NML;nil->CLIST;
        close;
COMMENT NML is copy of move list. CLIST is list of possible reverse moves;
LOOP:If MLIST.null  and PLOC.null and LOCFLG=1 then PRLOC(PLOCHD);.ASK;return;
        elseif MLIST.NULL and PLOC.NULL then .ASK;return;;
        elseif MLIST.NULL then goto LOC;
```

```
        close;
comment find which side moved,save no of captures(CP)
                and place move in CLIST,update board;
        BORW(MLIST.HD.HD)->BWL;CP->NC;MLIST.HD::CLIST->CLIST;
        nil->DDBF;[%[REVMOV],MLIST.HD,[R43EX]%]::DDB->DDB;

COMMENT update the board and list of pieces.i.e. make reverse move;
If BWL="B" then UPDATE(MLIST.HD,BLIST);else UPDATE(MLIST.HD,WLIST);close;

COMMENT last move known.it is worth looking here for the signifiant
                event  "is a king in check".
                if a king is not in check  then classify - module  B
                if both kings in check restore board and get next move;
LOOP1:ISKCHK(BLIST,WLIST)->CHI->KC;
If NOT(CHI=FALSE) and NOT(CHI.NULL) then [%[RETRO],CHI.HD,[R46EX]%]::DDB->DDB; goto L1;
elseif CHI.NULL then .CATPRB;close;;
L2:COPYALL(NBLIST)->BLIST;0->WHPFLG;COPYALL(NWLIST)->WLIST;.RSTB;TL(MLIST)->MLIST;
COMMENT a king is in check.how could this happen.reply in KL;NIL->CLIST;goto LOOP;
L1:     KCHECK1(CHI,KC)->KL;
        If not(KL.null) then [%[RETRO],KL,[R47EX]%]::DDB->DDB;close;
COMMENT if KL.null then no way found,else make moe & see if king
                still in check - indicating a capture;
If KL.NULL then [%[KCHECK],nil,[R42EX]%]::DDB->DDB; goto L2;
        elseif WHPFLG=1 then WHPCAP(CHI.HD.TL)->ZL;

                If ZL.TL.TL.null then 1->ILLFLG;goto L2;close;
                If ILLFLG=1 then goto L2;close;
        close;
        KL<>CLIST->CLIST;
NC-1->NC;If LOCFLG=0 then[%[KCHECK],KL,[R40EX]%]::DDB->DDB;close;
        If WHPFLG=1 then [%[WHPCAP],ZL,[R45EX]%]::DDB->DDB;goto L2;close;
COMMENT         if all reverse moves made print list;
If NOT(NC>1) then PRINTLS(CLIST);goto L2;
else BORW(KL.HD.HD)->BW;
If BW="B" then UPDATE(KL.HD,BLIST);else UPDATE(KL.HD,WLIST);close;
goto LOOP1;close;
END;



FUNCTION SCANTB;
COMMENT scan initial and final flags,print result of deduction;

        If BCFLGX>BCFLG and BCFLG=0 then nl(1);prstring('black cannot castle!);
        elseif not(BCAS.null) and BCFLGX/=0 then nl(1);PRSC(BCAS);
        close;
        If PPONBX>PPONB then nl(1);prstring('the promoted piece is on board!);nl(1);
        close;

        If WCFLGX>WCFLG and WCFLG=0 then nl(1);prstring('white cannot castle!);
        elseif not(WCAS.null) and WCFLGX/=0 then nl(1);PRSC(WCAS);
        close;

        If MSFLGX=1 and MSFLG=0 then .PRMP;close;
end;


FUNCTION PRSC XL;
COMMENT used by SCANTB .prints contents of BCAS/WCAS - castle tables;

vars    PC HL;BORW(XL.HD.HD.HD)->PC;
        nl(1);pr(PC);sp(1);prstring('can possibly castle with the following!);
        nl(1);XL.HD.HD->HL;HL=>;nl(1);
        If not(XL.TL.null) then XL.TL.HD.HD->HL;HL=>;nl(1);close;
```

```
end;
FUNCTION ASK;
        vars ITM DDBX HL;
COMMENT print result of deduction;
        .SCANTB;
COMMENT ask if explanation required;
        nl(2);prstring('explanation required <y or n>!);nl(1);
        .itemread->ITM;
        If ITM/="Y" then exit;
        COPYALL(DDB)->DDBX;REV(DDBX)->DDBX;

loop:   If DDBX.null then return;
        else dest(DDBX)->DDBX->HL;
                        HL.TL.HD;apply(valof(HL.TL.TL.HD.HD));goto loop;
        close;
end;

FUNCTION DPPROM=>CPSQ;
vars AH ALR;1->WHSQFL;NIL->DDBF;
COMMENT Checks through the pawn promotion rules.
                If a rule is triggered adds  (e.g) [WP XH YH] to CPSQ
                where XH,YH are the home coordinates;
nil->CPSQ;COPYALL(ALLRULES)->ALR;
COMMENT SETP sets up position variables;
loop:  If NOT(ALR.NULL) then dest(ALR)->ALR->AH;
        If category(valof(AH))="PROMRULE" and eval(premise(valof(AH)))
                then PROMXL<>CPSQ->CPSQ;close;
        goto loop;close;
        If not(CPSQ.NULL) then DDBF<>DDB->DDB;
        [%[DPPROM],[%CPSQ%],[R52EX]%]::DDB->DDB;CPSQ.HD.HD->PPAWN;
        If PPAWN="BP" then 1->BPPC;0->WPPC;
        elseif PPAWN="WP" then 1->WPPC;0->BPPC;
        close;
        close;
        0->WHSQFL;
end;
FUNCTION DRK XL;
IF XL.NULL THEN NIL;
ELSEIF XL.HD.HD="WR" OR XL.HD.HD="BR" OR XL.HD.HD="WK" OR XL.HD.HD="BK"
THEN DRK(TL(XL));
ELSE XL.HD::DRK(TL(XL));CLOSE;
END;


FUNCTION PRLOC XL;
nl(1);pr(XL.HD);sp(1);prstring('is on!);sp(1);pr("(");
pr(XL.TL.HD);pr(",");pr(XL.TL.TL.HD);pr(")");nl(1);
end;

FUNCTION BAMK KI XL=>NNL;
COMMENT a king is in check.if the checking piece moves is
                the same king still in check.if so implies capture.
                returns true or false;
vars    PC HHL R1 R2 ANS SF PCK;
        BORW(XL.HD.HD)->PC;nil->NNL;
loop:   If XL.null then exit;
        XL.HD->HHL;
        If PC="B" then UPDATE(HHL,BLIST);
        else UPDATE(HHL,WLIST);
        close;
        SFLAG->SF;ISKCHK(BLIST,WLIST)->R1->R2;
        If R1=false then goto L2;
        elseif R1.null then goto L1;
```

```
        elseif R2=KI and CAP="Y" and RVP(HHL) then 1->WHPFLG;
        elseif R2=KI and CAP="N" then goto L2;
        close;

L1:     HHL::NNL->NNL;
L2:     SF->SFLAG;RESTOR(HHL);XL.TL->XL;goto loop;
end;
```

```
   CCCCCCCCCCCC          AAAAAAAAA        TTTTTTTTTTTTTTT          111
   CCCCCCCCCCCC          AAAAAAAAA        TTTTTTTTTTTTTTT          111
   CCCCCCCCCCCC          AAAAAAAAA        TTTTTTTTTTTTTTT          111
CCC                   AAA         AAA          TTT             111111
CCC                   AAA         AAA          TTT             111111
CCC                   AAA         AAA          TTT             111111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
CCC                   AAA         AAA          TTT                111
   CCCCCCCCCCCC       AAA         AAA          TTT           111111111
   CCCCCCCCCCCC       AAA         AAA          TTT           111111111
   CCCCCCCCCCCC       AAA         AAA          TTT           111111111
```

```
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPP         PPP       OOO         OOO      PPP         PPP
PPP         PPP       OOO         OOO      PPP         PPP
PPP         PPP       OOO         OOO      PPP         PPP
PPP         PPP       OOO         OOO      PPP         PPP
PPP         PPP       OOO         OOO      PPP         PPP
PPPPPPPPPPPP          OOO         OOO      PPPPPPPPPPPP
PPPPPPPPPPPP          OOO         OOO      PPPPPPPPPPPP
PPPPPPPPPPPP          OOO         OOO      PPPPPPPPPPPP
PPP                   OOO         OOO      PPP
PPP                   OOO         OOO      PPP
PPP                   OOO         OOO      PPP
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                      OOOOOOOOO           PPP                 ......
PPP                      OOOOOOOOO           PPP                 ......
PPP                      OOOOOOOOO           PPP                 ......
```

```
*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>CAT1.POP.1, created:  4-Apr-84 12:36:15, printed: 11-Jun-86 10:
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION CAT1;
COMMENT it is possible to determine what pieces thepawns have captured;

        vars I J WL BL L1 L2 CL LT LX HXL HYL HHL BPR WPR TT BPRE WPRE;
        nil->WL;nil->BL;nil->BPR;nil->WPR;nil->BPRE;nil->WPRE;

COMMENT WL/BL contains [[[x,y]]wb wn]]    ]i.e. pieces captured
                by pawns on suare [x,y];

loop:   If not(CPW.null) then PLINXL(CPW.HD,"WP",WL)->WL;TL(CPW)->CPW;
        goto loop;
        close;

loop1:  If not(CPB.null) then PLINXL(CPB.HD,"BP",BL)->BL;TL(CPB)->CPB;
        goto loop1;
        close;

COMMENT now have pawn captures on 1st move,look for others;


        lengt(BDC)->LT;copyall(BDC)->LX;
        FORALL I 1 1 LT LX.HD->HL;HL.HD->HXL;HL.TL->HYL;TL(LX)->LX;
        [%HXL.HD,HYL.HD.HD,HYL.HD.TL.HD,HXL.TL.HD,HXL.TL.TL.HD%]->HHL;
        UPDATE(HHL,BLIST);HHL::BPR->BPR;
        close;

        FORALL I 1 1 8 FORALL J 2 1 5 If BOARD(I,J)="BP" and BOARD(I,7)="BP"
                then PLINXL([%I,J%],"BP",BL)->BL;GHSQ(I,J,"BP",BPR,BPRE)->BPRE->BPR;
        close;close;close;

        RESTB(NWLIST,NBLIST);

        lengt(WDC)->LT;copyall(WDC)->LX;
        FORALL I 1 1 LT LX.HD->HL;HL.HD->HXL;HL.TL->HYL;TL(LX)->LX;
        [%HXL.HD,HYL.HD.HD,HYL.HD.TL.HD,HXL.TL.HD,HXL.TL.TL.HD%]->HHL;
        UPDATE(HHL,WLIST);HHL::WPR->WPR;
        close;


        FORALL I 1 1 8 FORALL J 4 1 7 If BOARD(I,J)="WP" and BOARD(I,2)="WP"
                then PLINXL([%I,J%],"WP",WL)->WL;
                        GHSQ(I,J,"WP",WPR,WPRE)->WPRE->WPR;
        close;close;close;


COMMENT assumes at most 2 pawn captures;

        If lengt(WL)=2 then WL.HD.TL.HD->L1;WL.TL.HD.TL.HD->L2;
                If lengt(L1)>lengt(L2) then DBL(L1,L2)->WL.HD.TL.HD;
                elseif lengt(L2)>lengt(L1) then DBL(L2,L1)->WL.TL.HD.TL.HD;
                close;
        close;

        If lengt(BL)=2 then BL.HD.TL.HD->L1;BL.TL.HD.TL.HD->L2;
                If lengt(L1)>lengt(L2) then DBL(L1,L2)->BL.HD.TL.HD;
                elseif lengt(L2)>lengt(L1) then DBL(L2,L1)->BL.TL.HD.TL.HD;
                close;
        close;

        [%WL,BL%]->CL;[%[CAT1],CL,[R100EX]%]::DDB->DDB;RESTB(NWLIST,NBLIST);
        lengt(WL)+lengt(BL)->TT;CAT1A(BPR,WPR,BPRE,WPRE,TT,BL,WL);

end;
```

```
FUNCTION ADEND X XL=>XL;
COMMENT append x to xl;
        XL<>[%X%]->XL;
end;

FUNCTION PLINXL HL PWN XL=>XL;
COMMENT used in catl;

        vars ZL FL;
        WCAPPN(HL,PWN)->ZL;nil->FL;ZL::FL->FL;HL::FL->FL;FL::XL->XL;

end;


FUNCTION GHSQ I J PWN PRX PRE=>PRX PRE;
COMMENT all pawns that have captured on their first move
                are now on thir home squares.check pawn on (i,j);

        vars    XL X XH N;
        If PWN="WP" then WHTAB1->XL;-1->N;2->XH;
        else BHTAB1->XL;1->N;7->XH;
        close;

        GABS(I,XL)->X;[%PWN,I,J,X,J+N%]::PRX->PRX;
        If I-1/=XH then [%PWN,X,J+N,X,XH%]::PRX->PRX;
                        [%PWN,I,J,X,XH%]::PRE->PRE;
        close;

end;

FUNCTION RESTORA HL;
COMMENT modified version of restor (restore reverse move back
                to original position)
                allow for fact that in catla there may be 2 possible
                reverse moves;

        vars    X Y;
        HL.TL.TL.TL.HD->X;HL.TL.TL.TL.TL.HD->Y;
        If BOARD(X,Y)/=HL.HD then false;
        else true;
        close;

end;

FUNCTION CATlA BR WR BP WP TT BL WL;
COMMENT Called from catl to determine order in which pawns captured;


        vars     BPR WPR HL BCNT WCNT ML TOTCP POSC WLISTX BLISTX NM
                WLE BLE WC X1 X2 Y ZI LT LX HHL I HXL HYL HQ BPRE WPRE YL
                TMP TMPW TMPB BFLAG WFLAG LBL BCP WCP BH I;

BR->BPR;WR->WPR;TT->TOTCP;BP->BPRE;WP->WPRE;nil->YL;nil->TMPB;nil->TMPW;
nil->RKLST;0->BFLAG;0->WFLAG;

lengt(BL)->LBL;nil->BCP;
FORALL I 1 1 LBL BL.HD->BH;BH.TL.HD->BH;
        If not(BH.null) then BH<>BCP->BCP;close;
        TL(BL)->BL;close;

lengt(WL)->LBL;nil->WCP;
FORALL I 1 1 LBL WL.HD->BH;BH.TL.HD->BH;
        If not(BH.null) then BH<>WCP->WCP;close;
```

```
                TL(WL)->WL;close;

COMMENT update board froa all first capture moves;

        lengt(BDC)->LT;copyall(BDC)->LX;
        FORALL I 1 1 LT LX.HD->HL;HL.HD->HXL;HL.TL->HYL;TL(LX)->LX;
        [%HXL.HD,HYL.HD.HD,HYL.HD.TL.HD,HXL.TL.HD,HXL.TL.TL.HD%]->HHL;
        UPDATEX(HHL,BLIST,YL)->YL;
        close;

        lengt(WDC)->LT;copyall(WDC)->LX;
        FORALL I 1 1 LT LX.HD->HL;HL.HD->HXL;HL.TL->HYL;TL(LX)->LX;
        [%HXL.HD,HYL.HD.HD,HYL.HD.TL.HD,HXL.TL.HD,HXL.TL.TL.HD%]->HHL;
        UPDATEX(HHL,WLIST,YL)->YL;
        close;

COMMENT all first capture moves now on home squares
                check if any other pawns to be added to brp/wrp
                and to be placed on home square;

        lengt(WPRE)->LT;
        FORALL I 1 1 LT UPDATEX(WPRE.HD,WLIST,YL)->YL;TL(WPRE)->WPRE;close;

        lengt(BPRE)->LT;
        FORALL I 1 1 LT UPDATEX(BPRE.HD,BLIST,YL)->YL;TL(BPRE)->BPRE;close;


COMMENT         all pawns now on home suares,keep copy of wlist/blist
                to allow reset;

        copyall(WLIST)->WLISTX;copyall(BLIST)->BLISTX;
        lengt(BPR)->BLE;lengt(WPR)->WLE;0->POSC;nil->ML;0->BCNT;0->WCNT;

COMMENT look for possible move order;


L4:     BPR.HD->NM;TL(BPR)->BPR;BCNT+1->BCNT;

        If BCNT>BLE and POSC=TOTCP then .OUT;return;
        elseif BCNT>BLE and POSC/=TOTCP then RESTB(WLISTX,BLISTX);
        nil->TMPW;0->BCNT;0->WCNT;nil->ML;0->POSC;ADEND(NM,BPR)->BPR;nil->TMPB;
                0->WFLAG;0->BFLAG;goto L9;
        else NM.TL.HD->X1;NM.TL.TL.TL.HD->X2;NM.TL.TL.HD->Y;
        If not(RESTORA(NM)) then BCNT-1->BCNT;ADEND(NM,BPR)->BPR;
                goto L4;close;
        RESTORX(NM,YL);
        If X1=X2 then ADEND(NM,BPR)->BPR;1->BFLAG;goto L9;close;

COMMENT if pawns not capture on first move then cannot
                use wleft/bleft as possible captures,instead use bl/wl;

        .SETP;If BFLAG then BCP->WLEFT;LENGT(WLEFT)->WTOT;close; WCAPPN([%X1,Y%],'BP')->WC

                        If WC.null then ADEND(NM,BPR)->BPR;
                                goto L4;
                        close;

        DBL(WC,TMPB)->WC;WC.HD::TMPB->TMPB;HOMESQ(WC.HD,NM)->HQ;
        IF not(HQ.null) then [%WC.HD%]<>HQ->WC;
        else [%WC.HD%]->WC;
        close;
        If CAPPC/=0 and WC.HD=CAPPC then nl(1);prstring('the!);sp(1);
        pr(CAPPC);sp(1);prstring('was captured on!);sp(1);pr([%X1,Y%]);
        nl(1);
```

```
        close;

        [%NM%]<>[%WC%]->TMP;
        TMP::ML->ML;1+POSC->POSC;ADEND(NM,BPR)->BPR;
        close;

L9:     WPR.HD->NM;TL(WPR)->WPR;WCNT+1->WCNT;
        If WCNT>WLE and POSC=TOTCP then .OUT;return;
        elseif WCNT>WLE and POSC/=TOTCP then RESTB(WLISTX,BLISTX);nil->TMPB;
        0->BFLAG;0->WFLAG;
        0->BCNT;nil->TMPW;0->WCNT;nil->ML;0->POSC;ADEND(NM,WPR)->WPR;goto L4;
        else NM.TL.HD->X1;NM.TL.TL.TL.HD->X2;NM.TL.TL.HD->Y;
        If not(RESTORA(NM)) then WCNT-1->WCNT;ADEND(NM,WPR)->WPR;
                goto L9;close;
        RESTORX(NM,YL);
        If X1=X2 then ADEND(NM,WPR)->WPR;1->WFLAG;goto L4;close;
        .SETP;If WFLAG THEN WCP->BLEFT;LENGT(BLEFT)->BTOT;close; WCAPPN([%X1,Y%],"wp")->W
                        If WC.null then ADEND(NM,WPR)->WPR;
                                goto L9;
                        close;

        DBL(WC,TMPW)->WC;WC.HD::TMPW->TMPW;HOMESQ(WC.HD,NM)->HQ;
        If not(HQ.null) then [%WC.HD%]<>HQ->WC;
        else [%WC.HD%]->WC;
        close;

        if CAPPC/=0 and WC.HD=CAPPC then nl(1);prstring('the!);sp(1);

        pr(CAPPC);sp(1);prstring('was captured on!);sp(1);pr([%X1,Y%]);
        nl(1);
        close;

        [%NM%]<>[%WC%]->TMP;
        TMP::ML->ML;1+POSC->POSC;ADEND(NM,WPR)->WPR;goto L4;
        close;


end;

FUNCTION OUT;
        .CHKFLG;[%[CAT1A],ML,[R101EX]%]::DDB->DDB;
end;
```

```
   CCCCCCCCCCCC        AAAAAAAAA          SSSSSSSSSSS       RRRRRRRRRRR
   CCCCCCCCCCCC        AAAAAAAAA          SSSSSSSSSSS       RRRRRRRRRRRR
   CCCCCCCCCCCC        AAAAAAAAA          SSSSSSSSSSS       RRRRRRRRRRRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA      SSS                 RRR         RRR
CCC                 AAA       AAA         SSSSSSSS         RRRRRRRRRRR
CCC                 AAA       AAA         SSSSSSSS         RRRRRRRRRRRR
CCC                 AAA       AAA         SSSSSSSS         RRRRRRRRRRRR
CCC                 AAAAAAAAAAAAAAA             SSS       RRR   RRR
CCC                 AAAAAAAAAAAAAAA             SSS       RRR   RRR
CCC                 AAAAAAAAAAAAAAA             SSS       RRR   RRR
CCC                 AAA       AAA               SSS       RRR   RRR
CCC                 AAA       AAA               SSS       RRR   RRR
CCC                 AAA       AAA               SSS       RRR   RRR
   CCCCCCCCCCCC     AAA       AAA      SSSSSSSSSSS        RRR      RRR
   CCCCCCCCCCCC     AAA       AAA      SSSSSSSSSSS        RRR      RRR
   CCCCCCCCCCCC     AAA       AAA      SSSSSSSSSSS        RRR      RRR


PPPPPPPPPPP          OOOOOOOOO          PPPPPPPPPPP
PPPPPPPPPPPP         OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP         OOOOOOOOO          PPPPPPPPPPPP
PPP         PPP      OOO       OOO      PPP         PPP
PPP         PPP      OOO       OOO      PPP         PPP
PPP         PPP      OOO       OOO      PPP         PPP
PPP         PPP      OOO       OOO      PPP         PPP
PPP         PPP      OOO       OOO      PPP         PPP
PPPPPPPPPPPP         OOO       OOO      PPPPPPPPPPPP
PPPPPPPPPPPP         OOO       OOO      PPPPPPPPPPPP
PPPPPPPPPPPP         OOO       OOO      PPPPPPPPPPPP
PPP                  OOO       OOO      PPP                    ......
PPP                  OOO       OOO      PPP                    ......
PPP                  OOO       OOO      PPP                    ......
PPP                     OOOOOOOOO       PPP                    ......
PPP                     OOOOOOOOO       PPP                    ......
PPP                     OOOOOOOOO       PPP                    ......
```

```
*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>CASRUL.POP.1, created:  4-Apr-84 12:37:28, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION CASTLE;
COMMENT look through castle rules
                if e.g. have to consider 2 b castle positions bcflg=2
                if rule triggers bcflg-1->bcflg
                        rule explanation in ddb
                rules require:-
                        PSQS list of possible pawn promotion squares
                        PPIECE promoted piece [PP] or null
                        PPAWN promoted pawn "WP" or "BP"
                        PPOB =0 if promoted piece not on board,else 1
                        XYLIFS where promoted pawn crossed 7th or 2nd rank;
        vars    AH ALR;
        nil->CASXL;COPYALL(ALLRULES)->ALR;

loop:   If not(ALR.null) then dest(ALR)->ALR->AH;
                If category(valof(AH))="CASRULE" and eval(premise(valof(AH)))
                then [%[CASRULE],CASXL,explain(valof(AH))%]::DDB->DDB;
                close;
                nil->CASXL;goto loop;
        close;
end;

FUNCTION RULE30=>ANS;
COMMENT if a castle flag is set and   last move known and blist,wlist
                contains possible castle pieces,delete moves for these pieces;
        vars    XL KL R1 R2 ML MLH HL;
        false->ANS;

        If LM="B" and BCFLG/=0 then BLIST->XL;[BK 5 8]->KL;[BR 1 8]->R1;
                [BR 8 8]->R2;
        elseif LM="W" and WCFLG/=0 then WLIST->XL;[WK 5 1]->KL;[WR 1 1]->R1;
                [WR 8 1]->R2;
        else return;
        close;

        If not(membl(KL,XL)) then return;
        elseif not(membl(R1,XL)) and not(membl(R2,XL)) then return;
        close;
        COPYALL(MLIST)->ML;

loop:   If ML.null then return;
        else dest(ML)->ML->MLH;[%MLH.HD,MLH.TL.HD,MLH.TL.TL.HD%]->HL;
                If equal(KL,HL) or equal(R1,HL) or equal(R2,HL)
                then true->ANS;DELTL(MLH,MLIST)->MLIST;MLH::CASXL->CASXL;
                close;
        goto loop;
        close;
end;

FUNCTION RULE31;
COMMENT If a pawn has promoted on a square that contains a castle
                piece then this piece must have moved.
                PSQS contains list of promotion squares
                castle pieces in BCAS/WCAS e.g.[[[WR 1 1][WK 5 1]]];

        vars    XL PSQX YL PSQH;
        If PPAWN=0 then false ;return;
        elseif not(PPIECE.null) and PPIECE.HD="WR" then false;return;
        elseif not(PPIECE.null) and PPIECE.HD="BR" then false;return;
        elseif PPAWN="WP" and BCAS.null then false;return;
        elseif PPAWN="BP" and WCAS.null then false;return;
        elseif PPAWN="WP" then copyall(BCAS)->XL;
        else copyall(WCAS)->XL;
        close;
```

```
        COPYALL(PSQS)->PSQX;nil->YL;XL.HD<>YL->YL;
        If not(XL.TL.null) then XL.TL.HD<>YL->YL;close;

loop:   If PSQX.null then false;return;
        else dest(PSQX)->PSQX->PSQH;
                If equal(PSQH,YL.HD.TL) then YL.HD::CASXL->CASXL;true;.FLG1;
        DRCAS(YL.HD);return;
                elseif equal(PSQH,YL.TL.HD.TL) then YL.TL.HD::CASXL->CASXL;
                If PPAWN="WP" then nil->WCAS;else nil->BCAS;close;
                        true;.FLG1;return;
                elseif not(YL.TL.TL.null) and equal(PSQH,YL.TL.TL.HD.TL)
                then YL.TL.TL.HD::CASXL->CASXL;true;.FLG1;DRCAS(YL.TL.TL.HD);
                return;
                close;
        goto loop;
        close;
end;

FUNCTION RULE32=>ANS;
COMMENT if the promoted piece is a rook,constrained to its home square
                and the king moved to let it out,then no castle.
                promoted piece in PPIECE;

        vars P RNK XL YL XH QC BC RU RC KL RL RK R1 R2 FLG;
        false->ANS;nil->OMIT33;0->R32FLG;nil->CASXL;
        If PPIECE.null or PPOB=0 then return;
        else BORW(PPIECE.HD)->P;
                If P="W" then 2->RNK;"WR"->RK;WLIST->XL;FRMXL(WCAS)->YL;[WK 5 1]->KL;
                        [WR 1 1]->R1;[WR 8 1]->R2;
                else 7->RNK;"BR"->RK;BLIST->XL;FRMXL(BCAS)->YL;[BK 5 8]->KL;
                        [BR 1 8]->R1;[BR 8 8]->R2;
                close;
        close;
        If membl(R1,YL) and membl(R2,YL) then 1->FLG;
        else 0->FLG;
        close;


loop:   If XL.null then return;
        else dest(XL)->XL->XH;
                If XH.HD/=RK then goto loop;
                elseif XH.TL.TL.HD>RNK and XH.HD="WR" then goto loop;
                elseif XH.TL.TL.HD<RNK and XH.HD="BR" then goto loop;
                else GENCON(P)->QC->BC->RU->RC;FRMXL(RU)->RL;
                        If RU.null then goto loop;
                        elseif membl(XH,YL) and membl(XH,RL) and membl(KL,RL)
                        and FLG=1 then XH->OMIT33; goto loop;
                        elseif membl(XH,YL) and membl(XH,RL) and membl(KL,RL)
                        then DRCAS(XH);true->ANS;.FLG2;XH::CASXL->CASXL;
                        1->R32FLG;
                        elseif membl(KL,RL) and not(membl(XH,YL))
                        then true->ANS;XH::CASXL->CASXL;
                        close;
                close;
        goto loop;
        close;
end;

FUNCTION FRMXL XL=>YL;
COMMENT used by rule32;
        nil->YL;
loop:   If XL.null then return;
        else XL.HD<>YL->YL;TL(XL)->XL;goto loop;
```

```
        close;
end;


FUNCTION RULE33=>ANS;
COMMENT if the promoted piece is a rook and this rook is on a home
                square in a castle position then no castle;

        vars P CB RH1 RH2 XL;
        false->ANS;nil->CASXL;
        If PPIECE.null then return;
        elseif R32FLG=1 and PPIECE.HD="WR" and WCFLG>0 then return;
        elseif R32FLG=1 and PPIECE.HD="BR" and BCFLG>0 then return;
        else BORW(PPIECE.HD)->P;
                If P="W" then COUNTX("WR",PPIECE)->CB;[WR 1 1]->RH1;
                        [WR 8 1]->RH2;WLIST->XL;
                else COUNTX("BR",PPIECE)->CB;[BR 1 8]->RH1;[BR 8 8]->RH2;
                        BLIST->XL;
                close;
                        If CB=0 then exit;
        close;

        If equal(RH1,OMIT33) then nil->RH1;
        elseif equal(RH2,OMIT33) then nil->RH2;
        close;


If membl(RH1,XL) and ISRCAS(RH1) and RCCAS(RH1) then RH1::CASXL->CASXL;true->ANS;.FLG2;
        DRCAS(RH1);close;
If membl(RH2,XL) and ISRCAS(RH2) and RCCAS(RH2) then RH2::CASXL->CASXL;true->ANS;.FLG2;
        DRCAS(RH2);close;
end;
FUNCTION RULE34=>ANS;
COMMENT if a king moved to let out a rook that was captured on board
                then no castle.
                if 1 rook cap.on board and lengt(ru)=2 and k{ru then no castle.
                if 1 rook cap.on board and lengt(ru)=1 and k{ru and the rook
                in wcas/bcas not have same home square as rook in ru,no castle;

vars    XL RK HL CL P CB QC BC RC RU;

        If BCFLG/=0 then BLEFT->XL;"BR"->RK;[BK 5 8]->HL;copyall(BCAS)->CL;
        "B"->P;
        else WLEFT->XL;"WR"->RK;[WK 5 1]->HL;copyall(WCAS)->CL;"W"->P;
        close;

        false->ANS;COUNTX(RK,XL)->CB;GENCON(P)->QC->BC->RU->RC;

        If RU.null or CB=0 then return;
        elseif membl(HL,RU.HD) then goto L1;
        elseif not(RU.TL.null) and membl(HL,RU.TL.HD) then goto L1;
        else return;
        close;


L1:     If CB=1 and lengt(RU)=2 THEN true->ANS;HL->CASXL;
        elseif CB=1 and lengt(RU)=1 and NTSMR(CL,RU.HD) then true->ANS;
        HL->CASXL;
        else return;
        close;

        If BCFLG/=0 then BCFLG-1->BCFLG;
        else WCFLG-1->WCFLG;
        close;
```

```
        If P="B" then nil->BCAS;else nil->WCAS;close;
end;

FUNCTION NTSMR CL RU;
COMMENT cl=bcas/wcas,ru=list of rooks unconstrained but piece to move
                to let them out.
                if home square of rookin cl/= home square of rook in ru
                then true,else false;

vars    XL;
        RU.HD.HD->XL;
        If not(equal(XL,CL.HD.HD)) then true;
        elseif not(CL.TL.null) and not(equal(XL,CL.TL.HD)) then true;
        else false;
        close;
end;

FUNCTION RULE35=>ANS;
COMMENT if a pawn promoted such that a king was in check then no castle.
                list of squares where pawn crosed 2nd or 7th rank in XYLIFS;

        false->ANS;nil->CASXL;

        If PPAWN=0 then return;
        elseif PPAWN="WP" and BCAS.null then return;
        elseif PPAWN="BP" and WCAS.null then return;
        elseif PPAWN="WP" and membl([4 7],XYLIFS) then true->ANS;
                [4 7]::CASXL->CASXL;BCFLG-1->BCFLG;nil->BCAS;
        elseif PPAWN="WP" and membl([6 7],XYLIFS) then true->ANS;
                [6 7]::CASXL->CASXL;BCFLG-1->BCFLG;nil->BCAS;
        elseif PPAWN="BP" and membl([4 2],XYLIFS) then true->ANS;
                [4 2]::CASXL->CASXL;WCFLG-1->WCFLG;nil->WCAS;
        elseif PPAWN="BP" and membl([6 2],XYLIFS) then true->ANS;
                [6 2]::CASXL->CASXL;WCFLG-1->WCFLG;nil->WCAS;
        close;
end;

FUNCTION RULE36=>ANS;
COMMENT if a pawn captured a rook on its home square,then no castle
                list of rooks captured(from catla) in RKLIST;

        vars XL YL1 YL2 HL HHL P;

        copyall(RKLIST)->XL;FRMXL(WCAS)->YL1;FRMXL(BCAS)->YL2;false->ANS;
        nil->CASXL;

loop:   If XL.null then return;
        else XL.HD->HL;
                If membl(HL,YL1) or membl(HL,YL2) then DRCAS(HL);
                        true->ANS;HL::CASXL->CASXL;BORW(HL.HD)->P;.FLG2;
                close;
        TL(XL)->XL;goto loop;
        close;

end;

FUNCTION DRCAS XL;
COMMENT to delete XL ([WR 1 1]) from BCAS/WCAS;
        vars PC;
BORW(XL.HD)->PC;
        If PC="W" and equal(WCAS.HD.HD,XL) then WCAS.TL->WCAS;
        elseif PC="W" then [%WCAS.HD%]->WCAS;
        elseif PC="B" and equal(BCAS.HD.HD,XL) then BCAS.TL->BCAS;
```

```
        elseif PC="B"  then [%BCAS.HD%]->BCAS;
        close;

end;

FUNCTION FLG1;
        If PPAWN="WP" then BCFLG-1->BCFLG;
        else WCFLG-1->WCFLG;
        close;
        end;

FUNCTION FLG2;
        If P="W" then WCFLG-1->WCFLG;
        else BCFLG-1->BCFLG;
        close;
end;
```

```
FUNCTION RCCAS XL;
COMMENT used by rule 33 to check that the promoting pawn could
                actually reach the square;

        vars    QC BC RU RC P;
        BORW(PPIECE.HD)->P;GENCON(P)->QC->BC->RU->RC;

        If RC.null then true;
        elseif equal(XL,RC.HD.HD) then false;
        elseif RC.TL.null then true;
        elseif equal(XL,RC.TL.HD.HD) then false;
        else true;

        close;
end;
```

```
FUNCTION ISRCAS XL;
COMMENT to determine if xl ([wr xr yr]) is in the castle table;

        vars P CC;
        If XL.null then false;return;
        else BORW(XL.HD)->P;
                If P="W" then copyall(WCAS)->CL;
                else copyall(BCAS)->CL;
                close;
        close;

        If CL.null then false;return;
        elseif equal(XL,CL.HD.HD) then true;
        elseif not(CL.TL.null) and equal(XL,CL.TL.HD.HD) then true;
        else false;
        close;
end;
```

```
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHHHHHHHHHHHHHHH       EEEEEEEEEEEEE        CCC
CCC                     HHHHHHHHHHHHHHHH       EEEEEEEEEEEEE        CCC
CCC                     HHHHHHHHHHHHHHHH       EEEEEEEEEEEEE        CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
CCC                     HHH         HHH        EEE                  CCC
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC
    CCCCCCCCCCCC        HHH         HHH        EEEEEEEEEEEEEEE          CCCCCCCCCCCC


PPPPPPPPPPPP             OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP             OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP             OOOOOOOOO          PPPPPPPPPPPP
PPP         PPP        OOO         OOO      PPP         PPP
PPP         PPP        OOO         OOO      PPP         PPP
PPP         PPP        OOO         OOO      PPP         PPP
PPP         PPP        OOO         OOO      PPP         PPP
PPPPPPPPPPPP           OOO         OOO      PPPPPPPPPPPP
PPPPPPPPPPPP           OOO         OOO      PPPPPPPPPPPP
PPPPPPPPPPPP           OOO         OOO      PPPPPPPPPPPP
PPP                    OOO         OOO      PPP
PPP                    OOO         OOO      PPP
PPP                    OOO         OOO      PPP
PPP                    OOO         OOO      PPP                    ......
PPP                    OOO         OOO      PPP                    ......
PPP                    OOO         OOO      PPP                    ......
PPP                       OOOOOOOOO         PPP                    ......
PPP                       OOOOOOOOO         PPP                    ......
PPP                       OOOOOOOOO         PPP                    ......


*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>CHECK.POP.1, created: 16-Sep-83 13:17:58, printed: 11-Jun-86 10
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION ISKCHK BL WL=>KCH CLIST;
COMMENT is a king in check.if yes returns king in kch and e.g.
                [[wk bb 2 1]] in  clist.
                if k not in check clist is null.
                if 2 kings in check - illegal position - clist false;
VARS CLIST1 CLIST2 XBK YBK XWK YWK L1 L2;
FKING(BL,WL)->CLIST1;
CLIST1.HD.HD->XWK;
CLIST1.HD.TL.HD->YWK;
CLIST1.TL.HD.HD->XBK;
CLIST1.TL.HD.TL.HD->YBK;
BL.HD.HD->PC;
IF PC="BP" OR PC="BN" OR PC="BB" OR PC="BQ" OR PC="BK" OR PC="BR"
THEN BL->L1;WL->L2;
ELSE WL->L1;BL->L2;CLOSE;
CCHECK(L1,"WK",XWK,YWK)->CLIST1;CCHECK(L2,"BK",XBK,YBK)->CLIST2;
NIL->KCH;
IF NOT(NULL(CLIST1)) AND NOT(NULL(CLIST2)) THEN FALSE->CLIST;
ELSEIF NULL(CLIST1) AND NULL(CLIST2) THEN NIL->CLIST;
ELSEIF NOT(NULL(CLIST1)) THEN CLIST1->CLIST;
ELSE CLIST2->CLIST;CLOSE;
IF NOT(CLIST=FALSE) AND NOT(NULL(CLIST)) THEN CLIST.HD.HD->KCH;CLOSE;
END;
FUNCTION CCHECK XL CK X Y=>CLIST;
VARS HED XK YK ANS NLL;NIL->CLIST;X->XK;Y->YK;XL->NLL;
LOOP:IF NULL(NLL) THEN EXIT;
DEST(NLL)->NLL->HED;
CHK(HED,XK,YK)->ANS;IF ANS= FALSE THEN GOTO LOOP;
ELSE [%CK,HED.HD,HED.TL.HD,HED.TL.TL.HD%]::CLIST->CLIST;
GOTO LOOP;CLOSE;
END;
FUNCTION CHK XL XK YK;
VARS PC,XP,YP;XL.HD->PC;XL.TL.HD->XP;XL.TL.TL.HD->YP;
IF PC="WK" OR PC="BK" THEN FALSE;
ELSEIF PC="BB" OR PC="WB" THEN BISHOP(XP,YP,XK,YK);
ELSEIF PC="WN" OR PC="BN" THEN KNIGHT(XP,YP,XK,YK);
ELSEIF PC="WQ" OR PC="BQ" THEN QUEEN(XP,YP,XK,YK);
ELSEIF PC="WR" OR PC="BR" THEN ROOK(XP,YP,XK,YK);
ELSE PAWN(XP,YP,XK,YK);CLOSE;
END;
FUNCTION BISHOP XB YB XK YK;
IF NOT(XB+YB=XK+YK) AND NOT(YB-XB=YK-XK) THEN FALSE;EXIT;
IF XB+YB=XK+YK THEN GOTO B2;CLOSE;
B1:IF XB>=XK THEN YB-1->YB;XB-1->XB;
ELSE YB+1->YB;XB+1->XB;CLOSE;
IF(XB=XK) AND (YB=YK) THEN TRUE;RETURN;
ELSEIF NOT(BOARD(XB,YB)="BLANK") THEN FALSE;RETURN;
ELSE GOTO B1;CLOSE;
B2:IF XB>=XK THEN YB+1->YB;XB-1->XB;
ELSE YB-1->YB;XB+1->XB;CLOSE;
IF (XB=XK) AND (YB=YK) THEN TRUE;RETURN;
ELSEIF NOT(BOARD(XB,YB)="BLANK") THEN FALSE;RETURN;
ELSE GOTO B2;CLOSE;
END;
FUNCTION ROOK XR YR XK YK;
IF NOT(XR=XK) AND NOT (YR=YK) THEN FALSE;EXIT;
IF XR=XK THEN GOTO R2;CLOSE;
R1:IF XR<XK THEN XR+1->XR;
ELSE XR-1->XR;CLOSE;
IF (XR=XK) THEN TRUE;RETURN;
ELSEIF NOT(BOARD(XR,YR)="BLANK") THEN FALSE;RETURN;
ELSE GOTO R1;CLOSE;
R2:IF YR<YK THEN YR+1->YR;ELSE YR-1->YR;CLOSE;
IF YR=YK THEN TRUE;RETURN;
```

```
ELSEIF NOT(BOARD(XR,YR)="BLANK") THEN FALSE;RETURN;
ELSE GOTO R2;CLOSE;
END;
FUNCTION ABBS X;
IF X>0 THEN X;ELSE -X;CLOSE;
END;
FUNCTION KNIGHT XN YN XK YK;
VARS AB1 AB2;
ABBS(XN-XK)->AB1;ABBS(YN-YK)->AB2;
IF AB1>=3 OR AB2>=3 THEN FALSE;
ELSEIF AB1=1 AND AB2=2 THEN TRUE;
ELSEIF AB1=2 AND AB2=1 THEN TRUE;
ELSE FALSE;CLOSE;
END;
FUNCTION PAWN XP YP XK YK;
VARS AB;ABBS(XP-XK)->AB;
IF NOT(AB=1) THEN FALSE;
ELSEIF BOARD(XK,YK)="WK" AND (YP-YK)=1 THEN TRUE;
ELSEIF BOARD(XK,YK)="BK" AND (YK-YP)=1 THEN TRUE;
ELSE FALSE;CLOSE;
END;
FUNCTION QUEEN XQ YQ XK YK;
IF ROOK(XQ,YQ,XK,YK)=TRUE OR BISHOP(XQ,YQ,XK,YK)=TRUE THEN TRUE;
ELSE FALSE;CLOSE;
END;
FUNCTION FKING BL WL=>BW;
VARS PC S1 S2;BL.HD.HD->PC;NIL->BW;
IF PC="BP" OR PC ="BN" OR PC="BB" OR PC="BR" OR PC="BQ" OR PC="BK"
THEN GKING(BL,"BK")->S1;S1::BW->BW;GKING(WL,"WK")->S2;S2::BW->BW;
ELSE GKING(WL,"BK")->S1;S1::BW->BW;GKING(BL,"WK")->S2;S2::BW->BW;
CLOSE;
END;
FUNCTION GKING XL N;
IF XL.NULL THEN [%0,0%];
ELSEIF XL.HD.HD=N THEN [%XL.HD.TL.HD,XL.HD.TL.TL.HD%];
ELSE GKING(TL(XL),N);CLOSE;
END;

FUNCTION CHKLIN XL=>CLIN;
COMMENT a king is in check.function returns squares in checking line;
VARS CL XK YK XP YP HL PK;
NIL->CLIN;
IF XL.NULL THEN EXIT;
FKING(BLIST,WLIST)->CL;
IF XL.HD.HD="BK" THEN CL.TL.HD.HD->XK;CL.TL.HD.TL.HD->YK;
ELSE CL.HD.HD->XK;CL.HD.TL.HD->YK;CLOSE;
LOOP: IF XL.NULL THEN EXIT; DEST(XL)->XL->HL;HL.TL.HD->PC;HL.TL.TL.HD->XP;HL.TL.TL.TL.HD
                                                                          ->YP;
IF PC="BB" OR PC ="WB" THEN BICHK(XK,YK,XP,YP);
ELSEIF PC ="BR" OR PC = "WR" THEN RKCHK(XK,YK,XP,YP);
ELSEIF PC="BQ" OR PC = "WQ" THEN QUCHK(XK,YK,XP,YP);
ELSE GOTO LOOP;CLOSE;
->PK;PK<>CLIN->CLIN;GOTO LOOP;
END;
FUNCTION BICHK XK YK XP YP=>CL;
NIL->CL;VARS XX;ABBS(XP-XK)-1->XX;
IF XP<XK AND YP<YK THEN FORALL I 1 1 XX [%XP+I,YP+I%]::CL->CL;CLOSE;
ELSEIF XP>XK AND YP<YK THEN FORALL I 1 1 XX [%XP-I,YP+I%]::CL->CL;CLOSE;
ELSEIF XP<XK AND YP>YK THEN FORALL I 1 1 XX [%XP+I,YP-I%]::CL->CL;CLOSE;
ELSE FORALL I 1 1 XX [%XP-I,YP-I%]::CL->CL;CLOSE;CLOSE;
END;
FUNCTION RKCHK XK YK XP YP=>CL;
NIL->CL;VARS XX YY;ABBS(XP-XK)-1->XX;ABBS(YP-YK)-1->YY;
IF XP<XK THEN FORALL I 1 1 XX;[%XK-I,YK%]::CL->CL;CLOSE;
ELSEIF XP>XK THEN FORALL I 1 1 XX;[%XK+I,YK%]::CL->CL;CLOSE;
```

```
ELSEIF YP>YK THEN FORALL I 1 1 YY;[%XP,YK+I%]::CL->CL;CLOSE;
ELSE FORALL I 1 1 YY;[%XP,YK-I%]::CL->CL;CLOSE;CLOSE;
END;
FUNCTION QUCHK XK YK XP YP=>CL;
IF XP=XK OR YP=YK THEN RKCHK(XK,YK,XP,YP);
ELSE BICHK(XK,YK,XP,YP);CLOSE
->CL;
END;
```

```
   CCCCCCCCCCCC      LLL                 AAAAAAAAA           SSSSSSSSSSS
   CCCCCCCCCCCC      LLL                 AAAAAAAAA           SSSSSSSSSSS
   CCCCCCCCCCCC      LLL                 AAAAAAAAA           SSSSSSSSSSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA      SSS
CCC                  LLL                AAA       AAA         SSSSSSSSS
CCC                  LLL                AAA       AAA         SSSSSSSSS
CCC                  LLL                AAA       AAA         SSSSSSSSS
CCC                  LLL                AAAAAAAAAAAAAAA                  SSS
CCC                  LLL                AAAAAAAAAAAAAAA                  SSS
CCC                  LLL                AAAAAAAAAAAAAAA                  SSS
CCC                  LLL                AAA       AAA                    SSS
CCC                  LLL                AAA       AAA                    SSS
CCC                  LLL                AAA       AAA                    SSS
   CCCCCCCCCCCC      LLLLLLLLLLLLLLL    AAA       AAA      SSSSSSSSSSS
   CCCCCCCCCCCC      LLLLLLLLLLLLLLL    AAA       AAA      SSSSSSSSSSS
   CCCCCCCCCCCC      LLLLLLLLLLLLLLL    AAA       AAA      SSSSSSSSSSS


PPPPPPPPPPPP            OOOOOOOOO        PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO        PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO        PPPPPPPPPPPP
PPP         PPP       OOO       OOO      PPP         PPP
PPP         PPP       OOO       OOO      PPP         PPP
PPP         PPP       OOO       OOO      PPP         PPP
PPP         PPP       OOO       OOO      PPP         PPP
PPPPPPPPPPPP          OOO       OOO      PPPPPPPPPPPP
PPPPPPPPPPPP          OOO       OOO      PPPPPPPPPPPP
PPPPPPPPPPPP          OOO       OOO      PPPPPPPPPPPP
PPP                   OOO       OOO      PPP                    .....
PPP                   OOO       OOO      PPP                    .....
PPP                   OOO       OOO      PPP                    .....
PPP                   OOO       OOO      PPP                    .....
PPP                   OOO       OOO      PPP                    .....
PPP                   OOO       OOO      PPP                    .....
PPP                     OOOOOOOOO        PPP                    .....
PPP                     OOOOOOOOO        PPP                    .....
PPP                     OOOOOOOOO        PPP                    .....


*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>CLASS.POP.1, created: 30-Mar-84 13:19:59, printed: 11-Jun-86 10
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION CAT2;
COMMENT a pawn has promoted - on what square did it promote;

vars    PWN HHL XH YH BWL CL HL XL LFL ANS PRLSTX PRL XWLIST XBLIST;
        copyall(WLIST)->XWLIST;copyall(BLIST)->XBLIST;
        copyall(PPAWN)->PPCAP;nil->PRLSTX;0->AGC;If C3FLAG=1 then goto L1;close;
        PRLST.HD->HL;HL.HD->PWN;HL.TL.HD->XH;HL.TL.TL.HD->YH;[%XH,YH%]->PHSQ;
        copyall(PRLST)->PRLSTX;

COMMENT question1  on what square did the pawn promote;

        WHSQPP(XH,PWN)->XYLIFS->LIFC->PSQS;
        1->AGC;

COMMENT check if promoting pawn could actually capture;
        .CHCKPN;1+AGC->AGC;
        If not(.CHKFLG) then exit;


COMMENT question 2  is the promoted piece on board
                if yes  set ISPFLG;

        .ISPPB;1+AGC->AGC;If not(.CHKFLG) then exit;


COMMENT question 3  what is the promoted piece;

        .WISPP;1+AGC->AGC;If not(.CHKFLG) then exit;

        copyall(XYLIFS)->XYLIFX;
        If ISPFLG then goto L1;close;

COMMENT question 4
        it cannot be deduced that the promoted piece is on board
                can we find a square on which it was possibly captured;

If PPAWN="WP" and lengt(BDC)=1 then BDC->BWL;FCSQ(BDC)->HHL;BLIST->CL;
elseif PPAWN="WP" and lengt(WDC)=1 then WDC->BWL;FCSQ(WDC)->HHL;WLIST->CL;
else goto L1;
close;

[%[IPPRUL],HHL,[R64EX]%]::DDB->DDB;
nl(2);prstring('assuming the promoted piece is on board!);nl(1);


.SCANTB;nil->PPIECE;PPAWN->PPCAP;
nl(2);prstring('assuming the promoted piece was captured on !);sp(1);
HHL=>;nl(1);
.RFLGTAB;
        BWL.HD->HL;
        [%HL.HD.HD,HL.TL.HD.HD,HL.TL.HD.TL.HD,HL.HD.TL.HD,HL.HD.TL.TL.HD%]->XL;
        UPDATE(XL,CL);.SETP;

COMMENT question 5
        for each square on which the pawn promoted,can it be
                deduced that an opposition pawn promoted;

L1:     .DPPROM->PRLST;1+AGC->AGC;
        If not(PRLSTX.null) then DELTL(PRLSTX.HD,PRLST)->PRLST;close;
        lengt(PSQS)->LFL;lengt(PRLST)->PRL;If LFL=3 then 2->LFL;close;

        If PRL/=LFL then PPCAP->PPAWN;
COMMENT delete any entries from dpprom from ddb;
```

```
        FORALL I 1 1 PRL TL(DDB)->DDB;close;
        .PROMCP->ANS;

                If not(ANS) then 1->ILLFLG;false;exit;
        close;

        [%[DPPROM],nil,[R76EX]%]::DDB->DDB;
        PRLST.HD.HD->PPAWN;PRLST.HD->HL;HL.HD->PWN;HL.TL.HD->XH;HL.TL.TL.HD->YH;
        RESTB(XWLIST,XBLIST);.SETP;
        WHSQPP(XH,PWN)->XYLIFS->LIFC->PSQS;1+AGC->AGC;
        If not(.CHKFLG) then exit;
end;

FUNCTION CHCKPN;
        vars LLFC LIFCX YL;
        lengt(LIFC)->LLFC;copyall(LIFC)->LIFCX;nil->LIFC;
        If LIFCX.null then exit;
        FORALL I 1 1 LLFC WCAPPN(LIFCX.HD,PPAWN)->YL;
                If not(YL.null) then LIFCX.HD::LIFC->LIFC;close;
                TL(LIFCX)->LIFCX;
                close;

        If LIFC.null then 1->ILLFLG;
        else [%[CHCKPN],YL,[WCAPEX]%]::DDB->DDB;
        close;

end;

FUNCTION CAT3 CHI KC;
COMMENT category 3
                a king is in check.look through possible ways this could happen;

vars    KLX KLHX;
        [%[CAT3],CHI,[R82EX]%]::DDB->DDB;KCHECK1(CHI,KC)->KLX;

loop:   If not(KLX.null) then dest(KLX)->KLX->KLHX;
                [%[CAT3],KLHX,[R80EX]%]::DDB->DDB;CAT3A(KLHX);.RSTB;
                goto loop;
        close;

end;
FUNCTION CAT3A HL;
COMMENT slots in k in check frame;

vars    XF YF XT YT PN HHL;

        HL.HD->PN;HL.TL.HD->XF;HL.TL.TL.TL.HD->XT;HL.TL.TL.TL.TL.HD->YT;
        HL.TL.TL.HD->YF;

COMMENT slotl has a pawn promoted;

        If PN/="WP" and PN/="BP" then return;
        elseif YT/=8 and YT/=1 then return;
        close;

COMMENT a pawn has promoted - set up global variables;

        [%[%XF,YF%]%]->XYLIFS;[%[%XT,YT%]%]->PSQS;
        If XT/=XF then [%[%XT,YT%]%]->LIFC;
        else nil->LIFC;
        close;
```

```
        PN->PPAWN;
COMMENT a pawn has promoted,go to CAT2 control,set flag;
        [%[CAT3],nil,[R81EX]%]::DDB->DDB;1->C3FLAG;.CAT2;

end;

FUNCTION PROMCP=>ANS;
COMMENT used in CAT2
                if a pawn captured on promoting can it be deduced that for each
                piece captured an opposition pawn promoted;

        vars    X Y BLI WLI ZL XL HL YLH HHL PPN;


        false->ANS;
        If LIFC.null then return;
        else XYLIFS.HD.HD->X;XYLIFS.HD.TL.HD->Y;[%PPAWN,X,Y%]->ZL;
                copyall(BLIST)->BLI;copyall(WLIST)->WLI;copyall(LIFC)->XL;
                copyall(PPAWN)->PPN;
                        If PPAWN="WP" then ZL::WLIST->WLIST;
                        else ZL::BLIST->BLIST;
                close;
        close;

        PPAWN->BOARD(X,Y);.SETP;
COMMENT take each square on which the pawn captured in turn;

loop:   If XL.null then true->ANS;goto L1;
        else dest(XL)->XL->HL;WCAPPN(HL,PPCAP)->YL;
                If YL.null then goto L1;close;
        copyall(YL)->PPCOP;
        close;

        [%[PROMCP],YL,[WCAPEX]%]::DDB->DDB;
loop1:  If YL.null then goto loop;
        else dest(YL)->YL->YLH;YLH::HL->HHL;YLH->BOARD(HL.HD,HL.TL.HD);
                If PPAWN="WP" then HHL::BLIST->BLIST;
                else HHL:WLIST->WLIST;
                close;
        .DPPROM->PRLST;copyall(PPN)->PPAWN;
                If not(PRLST.null) then RESTB(WLI,BLI);
                goto loop1;
                close;
        close;

L1:     RESTB(NWLIST,NBLIST);.SETP;

end;

FUNCTION COLMP;
COMMENT what is the colour of the missing piece
                given square MISPQ [x y] try all b/w pieces inturn
                if can determine colour,place in MISCOL,else nil
                e.g.
                if all bp on board then use bleft
                elseif 1 bp missing (and promoted) use [bb br bq bn]
                        deleting those pieces in PPCOP
                else return

                if a pawn captured in promoting possible captured pieces in
                PPCOP
                try to eliminate pieces;

        vars    XL PC HL XM YM R1 R2 XXL BLT WLT;
```

```
        If MISPQ.null or MISCOL="WHITE" or MISCOL="BLACK" then exit;

        MISPQ.HD->XM;MISPQ.TL.HD->YM;nil->MISCOL;
        copyall(BLIST)->BLT;copyall(WLIST)->WLT;.RSTBO;

COMMENT try black first;

        If NPB=8 then BLEFT->XL;
        elseif PPCOP.null or NPB<7 then goto L1;
        else BORW(PPCOP.HD)->PC;
                If PC="W" then goto L1;
                close;
        [BB BN BQ BR]->XL;DBL(XL,PPCOP)->XL;copyall(XL)->XXL;
        close;

loop:   If not(XL.null) then dest(XL)->XL->HL;
                [%HL,XM,YM%]::BLIST->BLIST;HL->BOARD(XM,YM);
                ISKCHK(BLIST,WLIST)->R1->R2;
                        If not(R1) then D1(HL,XXL)->XXL;
                        close;
        "BLANK"->BOARD(XM,YM);TL(BLIST)->BLIST;goto loop;
        close;

        If XXL.null then "WHITE"->MISCOL;goto L2;close;

COMMENT now for white;

L1:     If NPW=8 then WLEFT->XL;
        elseif PPCOP.null or NPW<7 then goto L2;
        else BORW(PPCOP.HD)->PC;
                If PC="B" then goto L2;close;
        [WB WN WQ WR]->XL;DBL(XL,PPCOP)->XL;copyall(XL)->XXL;
        close;

loop1:  If not(XL.null) then dest(XL)->XL->HL;
                [%HL,XM,YM%]::WLIST->WLIST;HL->BOARD(XM,YM);
                ISKCHK(BLIST,WLIST)->R1->R2;
                        If not(R1)  then D1(HL,XXL)->XXL;
                        close;
        "BLANK"->BOARD(XM,YM);TL(WLIST)->WLIST;goto loop1;
        close;


        If XXL.null then "BLACK"->MISCOL;close;

COMMENT restore board to what it was on entry to routine;

L2:     RESTB(WLT,BLT);
end;

FUNCTION LFMP;
COMMENT look for missing piece - uses rule 90
                        - MISCOL;

        VARS    XL TOT YL SQ BI;

        If MISPQ.null then return;
        elseif MISCOL="WHITE" then BNUM->XL;WTOT->TOT;WLEFT->YL;"WB"->BI;
        elseif MISCOL="BLACK" then WNUM->XL;BTOT->TOT;BLEFT->YL;"BB"->BI;
        else return;
        close;

                If XL.HD/=TOT then return;
```

```
                elseif XL.TL.HD/=0 and XL.TL.TL.HD/=0 then return;
                else BICOL(YL)->SQ;
                        If SQ.null then return;
                        elseif SQ.HD=2 then BI->MISPC;0->MSFLG;
                                [%[LFMP],MISPC,[R90EX]%]::DDB->DDB;
                        close;
                close;

end;

FUNCTION PRMP;
COMMENT the missing piece is found,print it;

nl(1);prstring('the missing piece is a!);sp(1);pr(MISPC);nl(1);
end;
```

```
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW
CCC                 OOO         OOO     NNN         NNN     WWW         WWW
CCC                 OOO         OOO     NNN         NNN     WWW         WWW
CCC                 OOO         OOO     NNN         NNN     WWW         WWW
CCC                 OOO         OOO     NNNNNN      NNN     WWW         WWW
CCC                 OOO         OOO     NNNNNN      NNN     WWW         WWW
CCC                 OOO         OOO     NNNNNN      NNN     WWW         WWW
CCC                 OOO         OOO     NNN   NNN   NNN     WWW         WWW
CCC                 OOO         OOO     NNN   NNN   NNN     WWW         WWW
CCC                 OOO         OOO     NNN   NNN   NNN     WWW         WWW
CCC                 OOO         OOO     NNN     NNNNNN      WWW   WWW   WWW
CCC                 OOO         OOO     NNN     NNNNNN      WWW   WWW   WWW
CCC                 OOO         OOO     NNN     NNNNNN      WWW   WWW   WWW
CCC                 OOO         OOO     NNN       NNN       WWWWWW      WWWWWW
CCC                 OOO         OOO     NNN       NNN       WWWWWW      WWWWWW
CCC                 OOO         OOO     NNN       NNN       WWWWWW      WWWWWW
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW
    CCCCCCCCCCCC        OOOOOOOOO       NNN         NNN     WWW         WWW


PPPPPPPPPPPP        OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP        OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP        OOOOOOOOO       PPPPPPPPPPPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP                 ......
PPP                 OOO         OOO     PPP                 ......
PPP                 OOO         OOO     PPP                 ......
PPP                     OOOOOOOOO       PPP                 ......
PPP                     OOOOOOOOO       PPP                 ......
PPP                     OOOOOOOOO       PPP                 ......


*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>CONWP.POP.1, created: 10-Jun-86 16:31:40, printed: 11-Jun-86 11
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION DDBSET CL AH=>DDBF;
COMMENT place entries in dynamic data base DDB.;
[%[PROMRULE],CL.HD,explain(valof(AH))%]::DDBF->DDBF;
END;

FUNCTION RULE1=>ANS;
COMMENT Pawn promotion rule.If the b/w pawns have captured all
                  opposition pieces which can be captured on board(which include a pawn)
     and this pawn could not have reached a capture square;
        false->ANS;If PPAWN/=0 then exit;
vars CL PSQ;
nil->PSQ;
IF lengt(WLEFTP)=BPTT and NPW<8 then BTOT-WPTT->NCAP;
DRP1(CPBX,WHTAB,2,"WP")->CL;
        CPPROM(CL)->CL;
        If not(CL.null) then CL<>PSQ->PSQ;TRUE->ANS;
        DDBSET(CL,AH)->DDBF;close;close;
If lengt(BLEFTP)=WPTT and NPB<8 then WTOT-BPTT->NCAP;
DRP1(CPWX,BHTAB,7,"BP")->CL;CPPROM(CL)->CL;
        If not(CL.null) then CL<>PSQ->PSQ;TRUE->ANS;
        DDBSET(CL,AH)->DDBF;close;close;
COMMENT bptt/wptt = totasl pawn captures
                wtot/btot = number or pieces captured on board
                npw/npb = number of pawns on board
                now check that a pawn could actually promote;
If not(PSQ.null) then PSQ->PROMXL;close;
end;

FUNCTION RULE2=>ANS;
COMMENT Pawn promotion rule.If the b/w pawns have captured 8
                pieces and an opposition pawn could not have reached
                any of the capture squares;
        false->ANS;If PPAWN/=0 then exit;
vars PSQ CL;
nil->PSQ;
If BPTT=8 then BTOT-WPTT->NCAP;DRP1(CPWX,WHTAB,2,"WP")->CL;CPPROM(CL)->CL;
        If not(CL.null) then CL<>PSQ->PSQ;TRUE->ANS;DDBSET(CL,AH)->DDBF;close;close;
IF WPTT=8 then WTOT-BPTT->NCAP;DRP1(CPBX,BHTAB,7,"BP")->CL;CPPROM(CL)->CL;
        If not(CL.null) then CL<>PSQ->PSQ;TRUE->ANS;DDBSET(CL,AH)->DDBF;close;close;
If not(PSQ.NULL) then PSQ->PROMXL;close;
end;

FUNCTION RULE3=>ANS;
COMMENT Pawn promotion rule.If there are >2br/wr or >2wb/bb
                0r >2bn/wn or >1 bq/wq;
vars PSQ QCNT,RCNT NCNT CL;
nil->PSQ;FALSE->ANS;
CONTPC("WB")->BCNT;CONTPC("WQ")->QCNT;CONTPC("WR")->RCNT;CONTPC("WN")->NCNT;
If BCNT>2 or QCNT>1 or RCNT>2 or NCNT>2 then  [%[%"WP",WHTAB.HD,2%]%]->CL;
                TRUE->ANS;CL<>PSQ->PSQ;DDBSET(CL,AH)->DDBF;close;
CONTPC("BB")->BCNT;CONTPC("BQ")->QCNT;CONTPC("BR")->RCNT;CONTPC("BN")->NCNT;
If BCNT>2 or QCNT>1 or RCNT>2 or NCNT>2 then [%[%"BP",BHTAB.HD,7%]%]->CL;
                TRUE->ANS;CL<>PSQ->PSQ;DDBSET(CL,AH)->DDBF;close;
If not(PSQ.NULL) then PSQ->PROMXL;close;
end;

FUNCTION RULE4=>ANS;
COMMENT Pawn promotion rule.if one side has 2 bishops travelling on
                the same colour square;
vars XL PSQ LXL CL;
nil->PSQ;FALSE->ANS;
COMMENT obtain locations of all white bishops on board;
RELCB(WLIST)->XL;LENGT(XL)->LXL;
```

```
        If LXL>1 and CLSQT(XL.HD.HD,XL.HD.TL.HD,XL.TL.HD.HD,XL.TL.HD.TL.HD)
        then [%[%"WP",WHTAB.HD,2%]%]->CL;CL<>PSQ->PSQ;TRUE->ANS;DDBSET(CL,AH)->DDBF;close;
COMMENT         repeat for black bishops;
RELCB(BLIST)->XL;LENGT(XL)->LXL;

        If LXL>1 and CLSQT(XL.HD.HD,XL.HD.TL.HD,XL.TL.HD.HD,XL.TL.HD.TL.HD)
        then [%[%"BP",BHTAB.HD,7%]%]->CL;CL<>PSQ->PSQ;TRUE->ANS;DDBSET(CL,AH)->DDBF;close;
If not(PSQ.NULL) then PSQ->PROMXL;close;
end;


FUNCTION RULE5=>ANS;
COMMENT Pawn promotion rule.if one side has a bishop captured on its home
                square and there is a bishop on board travelling on the
                same colour square;
vars    XL YL HL X Y PSQ CL;
nil->PSQ;FALSE->ANS;
COMMENT obtain list of white bishops on board and list of white bishops
                captured at home;
RELCB(WLIST)->XL;WBHSQ->YL;
        If XL.NULL or YL.NULL then goto L1;close;
COMMENT         check if any bishops in XL on same colour square as
                squares given in YL;
loop:   If XL.NULL  then goto L1;
        else dest(XL)->XL->HL;HL.HD->X;HL.TL.HD->Y;
                If CLSQT(X,Y,YL.HD.HD,YL.HD.TL.HD)
                or not(YL.TL.NULL) and CLSQT(X,Y,YL.TL.HD.HD,YL.TL.HD.TL.HD)
                then TRUE->ANS;[%[%"WP",WHTAB.HD,2%]%]->CL;DDBSET(CL,AH)->DDBF;close;
                CL<>PSQ->PSQ;close;
        goto loop;close;
COMMENT repeat for black bishops;

L1:     RELCB(BLIST)->XL;BBHSQ->YL;
        If XL.NULL or YL.NULL then goto L2;close;
LOOP1:  If not(XL.NULL) then dest(XL)->XL->HL;
        HL.HD->X;HL.TL.HD->Y;
                If CLSQT(X,Y,YL.HD.HD,YL.HD.TL.HD)
                or not(YL.TL.NULL) and CLSQT(X,Y,YL.TL.HD.HD,YL.TL.HD.TL.HD)
                then TRUE->ANS;[%[%"BP",BHTAB.HD,7%]%]->CL;CL<>PSQ->PSQ;DDBSET(CL,AH)->
                                                                DDBF;close;
        goto loop1;close;
L2:     If not(PSQ.NULL) then PSQ->PROMXL;close;
end;


FUNCTION RULE6=>ANS;
COMMENT pawn promotion rule.if all pawn captures are on one colour
                and these captures include an oppostion pawn that could not
                have reached a capture square then this pawn promoted;

        false->ANS;If PPAWN/=0 then exit;
        If PPAWN=0 then exit;
vars    CL SQ X Y;
nil->PSQ;
COMMENT check for wp promotion;

        If BNUM.TL.HD/=0 and BNUM.TL.TL.HD/=0 then goto L1;
        elseif not(membl("WP",WLEFT)) then goto L1;
        elseif BNUM.TL.HD=0 then 1->X;1->Y;
        else 2->X;1->Y;
        close;
        BTOT-WPTT->NCAP;

        If BPTT=WTOT then DRP1(CPBX,WHTAB,2,"WP")->CL;CPPROM(CL)->CL;
                If not(CL.null) then CL<>PSQ->PSQ;true->ANS;
```

```
                DDBSET(CL,AH)->DDBF;
                close;
        elseif BPTT=WTOT-1 and membl("WB",WLEFT) then BICOL(WLEFT)->SQ;
                If not(SQ.null) and SQ.HD/=2 and CLSQT(X,Y,SQ.HD,SQ.TL.HD)
                then DRPl(CPBX,WHTAB,2,"WP")->CL;CPPROM(CL)->CL;
                        If not(CL.null) then CL<>PSQ->PSQ;true->ANS;
                        DDBSET(CL,AH)->DDBF;
                        close;
                close;
        close;

COMMENT  check for bp promotion;
        WTOT-BPTT->NCAP;

L1:     If WNUM.TL.HD/=0 and WNUM.TL.TL.HD/=0 then  goto L2;
        elseif not(membl("BP",BLEFT)) then return;
        elseif WNUM.TL.HD=0 then 1->X;1->Y;
        else 2->X;1->Y;
        close;

        If WPTT=BTOT then DRPl(CPWX,BHTAB,7,"BP")->CL;CPPROM(CL)->CL;
             If not(CL.null) then CL<>PSQ->PSQ;true->ANS;
             DDBSET(CL,AH)->DDBF;
             close;
        elseif WPTT=BTOT-1 and membl("BB",BLEFT) then BICOL(BLEFT)->SQ;
             If not(SQ.NULL) and SQ.HD/=2 and CLSQT(X,Y,SQ.HD,SQ.TL.HD)
             then DRPl(CPWX,BHTAB,7,"BP")->CL;CPPROM(CL)->CL;
                     If not(CL.null) then CL<>PSQ->PSQ;true->ANS;
                     DDBSET(CL,AH)->DDBF;
                     close;
             close;
        close;

L2:     If not(PSQ.null) then PSQ->PROMXL;close;

end;

FUNCTION RULE7=>ANS;
COMMENT pawn promotion rule.
                if a pawn captured on promoting and the only piece that it could
                have captured is an opposition pawn,then an opposing pawn
                promoted.

        vars    CL PSQ ;
        nil->PSQ;nil->CL;false->ANS;
        If LIFC.null then return;
        elseif PPCAP="WP" then BTOT-WPTT->NCAP;
        elseif PPCAP="BP" then WTOT-BPTT->NCAP;
        else return;
        close;

        If PPCAP="WP" and membl("BP",BLEFT) and NCAP=1
                THEN DRPl(CPWX,BHTAB,7,"BP")->CL;
        elseif membl("WP",WLEFT) and NCAP=1 then DRPl(CPBX,WHTAB,2,"WP")->CL;
        close;

        CPPROM(CL)->CL;
        If not(CL.NULL) then CL<>PSQ->PSQ;true->ANS;DDBSET(CL,AH)->DDBF;
        close;

        If not(PSQ.null) then PSQ->PROMXL;close;
end;

FUNCTION RULE8=>ANS;
```

```
COMMENT pawn promotion rule
                if a pawn made n captures in moving from its home square to a
                capture promotion square and  the only pieces available for
                captue are n opposition pawns then an opposition pawn promoted;

vars    CL PSQ XL YL PN LTAB RNK WB FLAG;

        nil->PSQ;nil->CL;false->ANS;0->FLAG;
        If LIFC.null then return;
        elseif PPCAP="WP" then BLEFT->XL;BTOT->YL;"BP"->PN;lengt(BHTAB)->LTAB;
                7->RNK;BHTAB->WB;
        elseif PPCAP="BP" then WLEFT->XL;WTOT->YL;"WP"->PN;lengt(WHTAB)->LTAB;
                2->RNK;WHTAB->WB;
        else return;
        close;

FORALL I 1 1 YL If XL.HD/=PN THEN 1->FLAG;close;TL(XL)->XL;close;
        If FLAG then exit;

FORALL I 1 1 LTAB [%PN,WB.HD,RNK%]::CL->CL;TL(WB)->WB;close;
        CPPROM(CL)->CL;
        If not(CL.null) then CL<>PSQ->PSQ;true->ANS;DDBSET(CL,AH)->DDBF;
        close;

        If not(PSQ.null) then PSQ->PROMXL;close;

end;

FUNCTION CPPROM XL=>YL;
COMMENT xl contains list of possible promotion pawns
                check that they could actually promote;

        vars    ZL L1 L2 L3 ZLH;
        COPYALL(XL)->ZL;nil->YL;

loop:   If ZL.null then 0->ILLFLG;
        else dest(ZL)->ZL->ZLH;WHSQPP(ZLH.TL.HD,ZLH.HD)->L1->L2->L3;
                If ILLFLG=1 then 0->ILLFLG;
                else ZLH::YL->YL;
                close;
        goto loop;
        close;
end;

FUNCTION DRP1 CPT WHT NM PWN=>CL;
VARS HT REP;NIL->CL;IF CPT.NULL THEN EXIT;WHT->HT;
LOOP:IF HT.NULL THEN RETURN;
ELSE PRO(HT.HD,CPT)->REP;
IF REP=TRUE THEN [%PWN,HT.HD,NM%]::CL->CL;CLOSE;
TL(HT)->HT;GOTO LOOP;CLOSE;
END;
FUNCTION CKPPR PS=>PPS;
VARS HL LS LC L;NIL->PPS;
LOOP:IF PS.NULL THEN RETURN;
ELSE DEST(PS)->PS->HL;WHSQPP(HL.TL.HD,HL.HD)->LS->LC->L;
IF NOT(L.NULL) THEN HL::PPS->PPS;CLOSE;
GOTO LOOP;CLOSE;
END;
FUNCTION GSQ XL=>SQU;
VARS CL;IF P="W" THEN [[WQ 4 1]]->CL;ELSE [[BQ 4 8]]->CL;CLOSE;
LOOP:IF XL=CL.HD.HD THEN [%CL.HD.TL.HD,CL.HD.TL.TL.HD%]->SQU;
ELSE TL(CL)->CL;GOTO LOOP;CLOSE;
END;
FUNCTION CLSQT XP YP XB YB;
```

```
IF ERASE((XB+YB)//2)=ERASE((XP+YP)//2) THEN TRUE;ELSE FALSE;CLOSE;
END;
FUNCTION PRO HL CPT;
VARS XH HHL;GABS(HL,CPT)->XH;ABBS(XH-HL)->HHL;
IF HHL>4 OR HHL>NCAP THEN TRUE;ELSE FALSE;CLOSE;
END;
FUNCTION CSQ1 XL=>OUT;
VARS HL;NIL->OUT;
LOOP:IF XL.NULL THEN EXIT;XL.HD.TL->HL;TL(XL)->XL;
LOOP1:IF HL.NULL THEN GOTO LOOP;
ELSE HL.HD::OUT->OUT;TL(HL)->HL;GOTO LOOP1;CLOSE;
END;
FUNCTION CSQ XL=>OUT;
VARS HL;NIL->OUT;
LOOP:IF XL.NULL THEN EXIT;XL.HD.TL->HL;TL(XL)->XL;
LOOP1:IF HL.NULL THEN GOTO LOOP;
ELSE HL.HD.HD::OUT->OUT;TL(HL)->HL;GOTO LOOP1;CLOSE;
END;
FUNCTION DELTL X XL;
IF XL.NULL THEN NIL;
ELSEIF EQUAL(X,XL.HD) THEN DELTL(X,TL(XL));
ELSE XL.HD::DELTL(X,TL(XL));CLOSE;
END;
FUNCTION CKPMS PWN LIF LIFC=>LIF LIFC;
VARS BI BSQ LOCB XP YP ML HL ANS HHL XXL PS YL;
IF PWN="BP" THEN "BB"->BI;BBHSQ->BSQ;RELCB(BLIST)->LOCB;
ELSE "WB"->BI;WBHSQ->BSQ;RELCB(WLIST)->LOCB;CLOSE;
MEMLST(PSQQ,BSQ)->ANS;IF LOCB.NULL OR BSQ.NULL OR ANS=TRUE THEN EXIT;
LIF->XXL;LOCB->YL;
LOOP:IF XXL.NULL THEN EXIT;DEST(XXL)->XXL->HL;HL.HD->XP;HL.TL.HD->YP;
[%XP,YP%]->PS;PCMOV([%[%BI,XP,YP%]%])->ML;CLSQT(PSQQ.HD,PSQQ.TL.HD,XP,YP)->ANS;
IF ANS=FALSE OR NOT(ML.NULL) THEN GOTO LOOP;CLOSE;
LOOP1:IF LOCB.NULL THEN YL->LOCB;GOTO LOOP;CLOSE;
DEST(LOCB)->LOCB->HHL;CLSQT(HHL.HD,HHL.TL.HD,XP,YP)->ANS;
IF ANS=TRUE THEN PRCK(PS,PWN);DELTL(PS,LIF)->LIF;DELTL(PS,LIFC)->LIFC;CLOSE;
GOTO LOOP;
END;
FUNCTION WHSQPP XH PWN=>LIF LIFC LIFS;
VARS NX RNK HF PN NC ADD CNT ANS KI REP HL HHL CPL BI XS YS CB ML XN YSQ LFC;
        vars LIFSX NCP PPX X1;

NIL->LIF;0->ILLFLG;NIL->LIFC;NIL->LIFS;

IF PWN="BP" THEN 2->RNK;3->YSQ;WTOT-BNUM.HD->NCAP;WTOT->XN;WLEFT->CPL;"WB"->BI;1->HF;
ELSE 7->RNK;6->YSQ;BTOT-WNUM.HD->NCAP;BTOT->XN;BLEFT->CPL;"BB"->BI;8->HF;"BP"->PN;
0->NC;XH->NX;1->ADD;0->CNT;LFK(PNM)->REP;
LOOP:IF BOARD(NX,RNK)/=PN AND NCAP=NC THEN .LFK1->ANS;
IF ANS=TRUE THEN  [%NX,HF%]::LIF->LIF;[%NX,RNK%]::LIFS->LIFS;
ELSE GOTO L1;CLOSE;
ELSEIF BOARD(NX,RNK)/=PN AND NC<NCAP THEN .LFK1->ANS;
IF ANS=TRUE THEN [%[%NX-1,HF%],[%NX,HF%],[%NX+1,HF%]%]<>LIF->LIF;
[%[%NX-1,HF%],[%NX+1,HF%]%]<>LIFC->LIFC;[%NX,RNK%]::LIFS->LIFS;CLOSE;
CLOSE;
1+NC->NC;
L2:IF NOT(NC>NCAP) THEN NX+ADD->NX;
IF NOT(NX>8) AND NOT(NX<1) THEN GOTO LOOP;CLOSE;CLOSE;
L1:1+CNT->CNT;
IF CNT/=2 AND NCAP/=0 THEN XH->NX;-1->ADD;1->NC;GOTO L2;
ELSE SHUFF(LIF)->LIF;SHUFF(LIFC)->LIFC;CLOSE;
IF NOT(REP.NULL) THEN DELTL(REP,LIF)->LIF;DELTL(REP,LIFC)->LIFC;CLOSE;
        If LIFS.null or LIF.null then 1->ILLFLG;exit;
LIFS.HD.HD->XS;LIFS.HD.TL.HD->YS;CNPWN(BI,CPL)->CB;
PCMOV([%[%PWN,XS,YS%]%])->ML;LEGCHK(CAP,ML)->ML;
DCSQ(ML,LIF,LIFC)->LIFC->LIF;CHKLLC(LIFC,LIF)->LIF->LIFC;
```

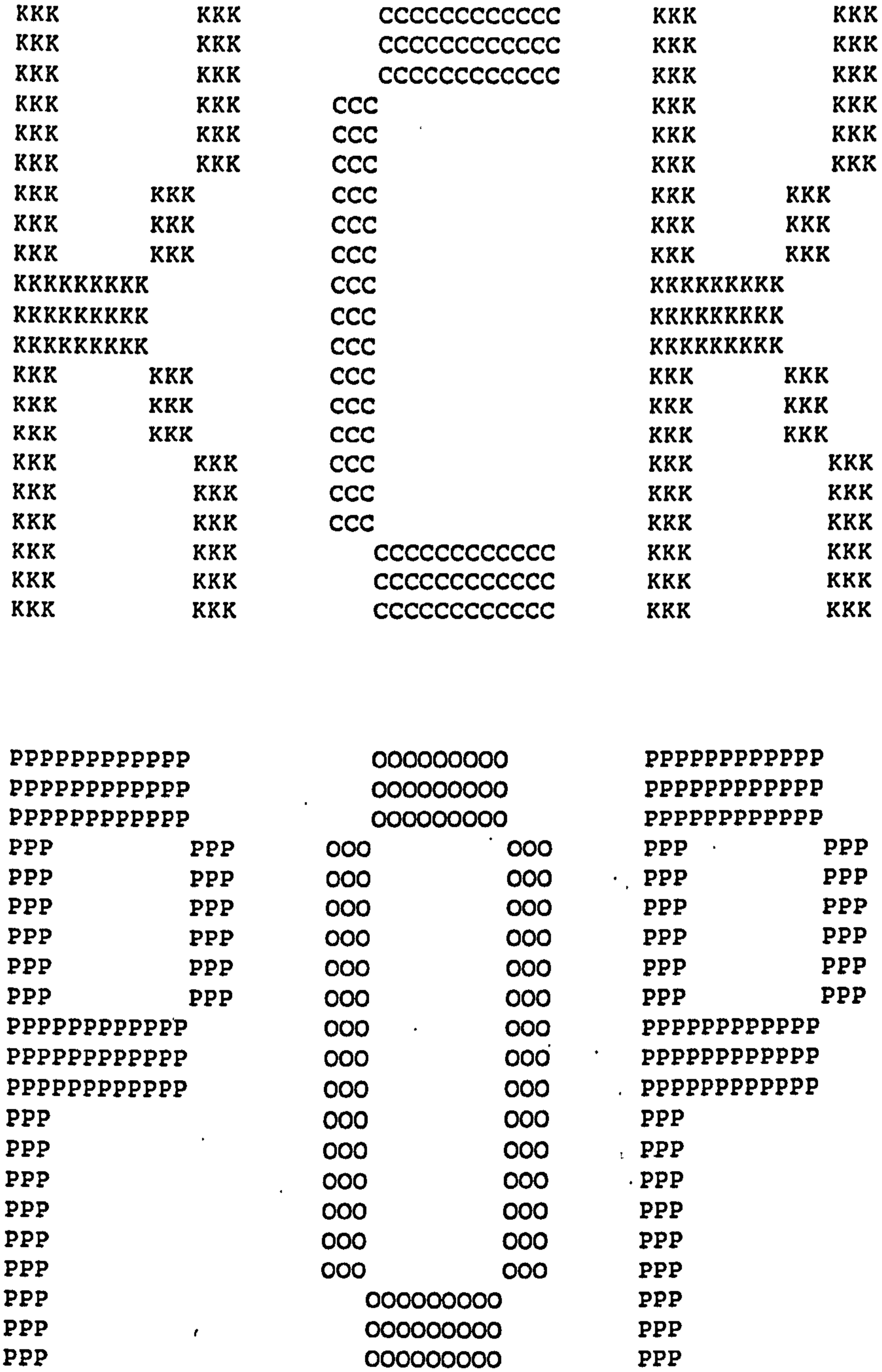
```
CKPMS(PWN,LIF,LIFC)->LIFC->LIF;


COMMENT         if (say) a wp promoted with home square (x,y)
                and ith sa been deduced that a bp promoted  cossing
                the square (x,y) and (x,y) is on the same file as the
                bp and the wp could not make any captures
                then the bp could not have crossed on this square;

        If PPCAP="WP" then BTOT-WPTT->NCP;
        elseif PPCAP="BP" then WTOT-BPTT->NCP;
        else goto L3;
        close;

        lengt(LIFS)->LFL;copyall(LIFS)->LIFSX;
loop2:  IF LIFSX.null then goto L3;
        elseif equal(LIFSX.HD,PHSQ) and NCP<2 then DELTL(LIFSX.HD,LIFS)->LIFS;
        close;
        TL(LIFSX)->LIFSX;goto loop2;
L3:     If LIFS.null or LIF.null then 1->ILLFLG;exit;

COMMENT if lengt(lifs)=1 and the pawn promoted in moving to lifs
                then add this capture to bnum/wnum;

        If lengt(LIFS)=1 then LIFS.HD.HD->X1;
                If PWN="BP" and BOARD(X1,3)="WP" then ALTNUM(X1,3,BNUM)->BNUM;
                elseif BOARD(X1,6)="BP" then ALTNUM(X1,6,WNUM)->WNUM;
                close;
        close;
IF NCAP=1 AND LENGT(LIFS)=1 AND ABBS(XH-LIFS.HD.HD)=1 THEN NIL->LIFC;[%[%LIFS.HD.HD,LIF.
                                                                    .HD,TL.HD%],]=>LIF;
CLOSE;


COMMENT place lists in DDB;

        If WHSQFL=0 then
[%[WHSQPP],[%LIF,LIFC,LIFS%],[R50EX]%]::DDB->DDB;
        close;
COMMENT         apply R75EX;

        If lengt(LIFS)=1 and BOARD(LIFS.HD.HD,YSQ)=PN and not(LIFC.null)
        and lengt(CPL)=2 and  CPL.HD=BI and CPL.TL.HD=BI
        and ABBS(LIFS.HD.HD-XH)=1
        then lengt(LIFC)->LFC;[%[WHSQPP],LIFC,[R75EX]%]::DDB->DDB;
                FORALL I 1 1 LFC DELTL(LIFC.HD,LIF)->LIF;TL(LIFC)->LIFC;close;
                nil->LIFC;
        close;

end;


FUNCTION CHKLLC LIFC LIF=>LIFC LIF;
VARS XL HL ANS PC QC BC RU RC CPSQ;
PNM->XL;NIL->CPSQ;
LOOP:IF NOT(XL.NULL) THEN DEST(XL)->XL->HL;TL(HL)::CPSQ->CPSQ;GOTO LOOP;CLOSE;
BORW(PWN)->PC;IF PC="W" THEN "B"->PC;ELSE "W"->PC;CLOSE;GENCON(PC)->QC->BC->RU->RC;
LOOP1:IF NOT(RC.NULL) THEN RC.HD.TL->XL;
IF LENGT(XL)=1 THEN XL.HD::CPSQ->CPSQ;CLOSE;
TL(RC)->RC;GOTO LOOP1;CLOSE;
LOOP2:IF NOT(BC.NULL) THEN BC.HD.TL<>CPSQ->CPSQ;TL(BC)->BC;GOTO LOOP2;CLOSE;
IF NOT(QC.NULL) THEN QC.HD.TL<>CPSQ->CPSQ;CLOSE;
DBLL(LIFC,CPSQ)->LIFC;DBLL(LIF,CPSQ)->LIF;
END;

FUNCTION ALTNUM X Y XL=>XL;
COMMENT used in WHSQPP
```

```
                ADD PAWN CAPTURES TO BNUM/WNUM;

        vars A B C;
        XL.HD->A;XL.TL.HD->B;XL.TL.TL.HD->C;
        1+A->A;If CLSQT(1,1,X,Y) then 1+B->B;else 1+C->C;close;
        [%A,B,C%]->XL;
end;

FUNCTION DBLL XXL XL=>YL;
XXL->YL;
LOOP:IF XL.NULL THEN RETURN;ELSE DELTL(XL.HD,YL)->YL;
TL(XL)->XL;GOTO LOOP;CLOSE;
END;
FUNCTION DCSQ ML LIF LIFC=>XF XC;
VARS HL LH XL;LIF->XL;COPYALL(LIF)->XF;COPYALL(LIFC)->XC;
LOOP:IF ML.NULL THEN RETURN;
ELSE ML.HD->HL;IF HL.TL.HD=HL.TL.TL.TL.HD THEN TL(ML)->ML;GOTO LOOP;
ELSE XL->LIF;CLOSE;CLOSE;
LOOP1:IF LIF.NULL THEN TL(ML)->ML; GOTO LOOP;
ELSE LIF.HD->LH;IF XS/=LH.HD AND CB=2 AND XN=2 THEN DELTL(LH,XF)->XF;DELTL(LH,XC)->XC;
ELSEIF XS/=LH.HD AND XN=1 THEN DELTL(LH,XF)->XF;DELTL(LH,XC)->XC;CLOSE;
TL(LIF)->LIF;GOTO LOOP1;CLOSE;
END;
FUNCTION LFK1;
VARS YL;[%NX,RNK%]->YL;
IF REP.NULL THEN TRUE;
ELSEIF EQUAL(YL,HL) OR EQUAL(YL,HHL) THEN FALSE;
ELSE TRUE;CLOSE;
END;
FUNCTION LFK XL;
IF XL.NULL THEN NIL;
ELSEIF XL.HD.HD=KI THEN [%XL.HD.TL.HD,XL.HD.TL.TL.HD%];
ELSE LFK(TL(XL));CLOSE;
END;
FUNCTION SHUFF XL=>LST;
VARS ANS;NIL->LST;ELI(XL)->XL;
LOOP:IF XL.NULL THEN EXIT;MEMLST(XL.HD,TL(XL))->ANS;
IF ANS=TRUE THEN XL.HD::LST->LST;CLOSE;TL(XL)->XL;GOTO LOOP;
END;
FUNCTION ELI XL;
IF XL.NULL THEN NIL;
ELSEIF XL.HD.HD>8 OR XL.HD.HD<1 THEN ELI(TL(XL));
ELSE XL.HD::ELI(TL(XL));CLOSE;
END;
FUNCTION BICOL XL=>OUT;
VARS CB PC BL BI;0->CB;COUNTB(XL,CB)->CB;
IF CB=2 THEN [2]->OUT;RETURN;ELSEIF CB=0 THEN NIL->OUT;RETURN;CLOSE;
BORW(XL.HD)->PC;
IF PC="B" THEN LFORB(BLIST)->BL;"BB"->BI;
ELSE LFORB(WLIST)->BL;"WB"->BI;CLOSE;
IF BL.NULL THEN LCONS(BI)->OUT;
ELSE BL->OUT;CLOSE;
END;
FUNCTION LCONS BI;
IF BI="BB" AND BOARD(2,7)="BP" AND BOARD(4,7)="BP" THEN [3 8];
ELSEIF BI="BB" AND BOARD(5,7)="BP" AND BOARD(7,7)="BP" THEN [6 8]
ELSEIF BI="WB" AND BOARD(2,2)="WP" AND BOARD(4,2)="WP" THEN [3 1];
ELSEIF BI="WB" AND BOARD(5,2)="WP" AND BOARD(7,2)="WP" THEN [6 1];
ELSE NIL;
CLOSE;
END;
```

```
FUNCTION  PLACE XL;
IF NOT(XL.NULL)THEN XL.HD.HD->BOARD(XL.HD.TL.HD,XL.HD.TL.TL.HD);
PLACE(TL(XL));
CLOSE;
END;
FUNCTION DISPLAY;
VARS ROW COL;1->COL;8->ROW;NL(1);
LOOP: PRN(BOARD(COL,ROW));
      IF COL<8 THEN COL+1->COL;GOTO LOOP
       ELSEIF ROW=1 THEN NL(1);
         ELSE 1->COL;ROW-1->ROW;NL(1);GOTO LOOP
       CLOSE;
END;
FUNCTION PRN SQUARE;
IF SQUARE="BLANK" THEN SP(2); PR(".");
 ELSE PR(SQUARE);PR(".");
CLOSE;
END;
```

```
FUNCTION EVAL LIST;
VARS FN,X;
VALOF(HD(LIST))->FN;
TL(LIST)->LIST;
REP:IF NULL(LIST) THEN GOTO QUIT CLOSE;
HD(LIST)->X;
TL(LIST)->LIST;
IF ATOM(X) THEN X ELSE EVAL(X) CLOSE;
GOTO REP;
QUIT:
APPLY(FN);
END;
```

```
1->FULLERR;1->NICERR;
MACRO POPEXIT; [EXIT].POPMESS;END;
MACRO READIN;
  VARS FILENAME;
  .ITEMREAD->FILENAME;
  MACRESULTS([%
    "POPMESS","(","[","COMPILE ",
      FILENAME,
     ".","POP",
    "]",")",";"
            %])
END;
[WELCOME TO POP2]=>
```



START Job BAGAT Req #751 for O.ENABLEDOP Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>KCK.POP.1, created: 5-Aug-83 14:36:02, printed: 11-Jun-86 11:0
Job parameters: Request created:11-Jun-86 10:53:56 Page limit:225 Forms:XERO
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:

```
FUNCTION KCHECK1 CHI KC=>PL;
        vars    ML CX;
        If CAP=0 then "Y"->CAP;close;
        If LM="U" then nil->ML;else MLIST->ML;close;
        CHKLIN(CHI)->CX;KCHECK(ML,KC,CX,CHI,BLEFT,WLEFT,CAP)->PL;
end;

FUNCTION KCHECK XL KC CL CHL BC WC CAP=>KLIST;
VARS XXL;
NIL->KLIST;
CANCHK(KC,CHL,CAP)->XXL;
IF NOT(XXL.NULL) THEN XXL<>KLIST->KLIST;CLOSE;
IF CL.NULL THEN EXIT;
MOVCHK(XL,KC,CL,CHL,BC,WC)->XXL;
IF NOT(XXL.NULL) THEN XXL<>KLIST->KLIST;CLOSE;
MOVLIN(KC,CL,CHL)->XXL;
IF NOT(XXL.NULL) THEN XXL<>KLIST->KLIST;CLOSE;
PROM(CL,KC)->XXL;IF NOT(XXL.NULL) THEN XXL<>KLIST->KLIST;CLOSE;

COMMENT         look for any looping.see e.g p 145 where a bk ins in check
                and only way for htis to happen is for a wn moving from
                check line.same k also in check from another piece and only
                way for this to happen is for same wn to move;
        If not(KLIST.null) then BAMK(KC,KLIST)->KLIST;close;
COMMENT this routine set flag WHPFLG if an oppo piece can
                be captured.;
END;
FUNCTION CANCHK KC CHL CAP=>CANLST;

COMMENT can the checking piece be moved to the check position
                other than along the checking line;
VARS XL R1 R2 PC CH DH PLIST ML ANS BXX LEGX;COPYALL(MLIST)->ML;
nil->CANLST;[%[%CHL.HD.TL.HD,CHL.HD.TL.TL.HD,CHL.HD.TL.TL.TL.HD%]%]->XL;
COMMENT before using subset of legality checker,save variables;
        COPYALL(DDBFX)->BXX;COPYALL(LEGDEL)->LEGX;
        PCMOV(XL)->MLIST;COPYALL(MLIST)->PLIST;.RULE13->ANS;COPYALL(ML)->MLIST;
COMMENT restore variables;
        BORW(XL.HD.HD)->PC;COPYALL(LEGX)->LEGDEL;COPYALL(BXX)->DDBFX;
        If PC="B" then BLIST->CH;WLIST->DH;
        else WLIST->CH;BLIST->DH;
        close;
LOOP:   If PLIST.null then exit;
COMMENT update board and see if king in check;
        UPDATE(PLIST.HD,CH);ISKCHK(CH,DH)->R1->R2;
        If NOT(R1=FALSE) and (R2=KC) and NOT(R1.HD.TL.TL.HD=PLIST.HD.TL.TL.TL.HD
        and NOT(R1.HD.TL.TL.TL.HD=PLIST.HD.TL.TL.TL.TL.HD) then PLIST.HD::CANLST->CANLST;
        close;
        RESTOR(PLIST.HD);TL(PLIST)->PLIST;GOTO LOOP;
END;

FUNCTION MOVCHK XL KC CL CHL BC WC => MLX;
VARS HML CHLOC PC   ;NIL->MLX;
COMMENT could a none checking piece have moved from the checking line
                to discover check and then be captured by the king;
        DIDKM(XL,KC) -> HML;
COMMENT if the king did not move then exit;
        If HML.null then exit;
        nil -> CHLOC; CHL.HD.TL.HD -> PC;
        [%CL.HD.HD,CL.HD.TL.HD%]::CHLOC -> CHLOC;
        If NOT(NULL(CL.TL)) then [%CL.TL.HD.HD,CL.TL.HD.TL.HD%]::CHLOC -> CHLOC;
        close;
COMMENT goto subroutine,depending on piece;
```

```
        If PC="BB" or PC="WB" then BCHK(CHLOC,HML,KC,BC,WC) -> MLX;
        elseif PC="BR" or PC="WR" then RCHK(CHLOC,HML,KC,BC,WC) -> MLX;
        elseif PC="BQ" or PC="WQ" then QCHK(CHLOC,HML,KC,BC,WC) -> MLX;
        close;
end;

FUNCTION DIDKM LI KI;
        If LI.NULL then NIL;
        elseif LI.HD.HD=KI then LI.HD::DIDKM(TL(LI),KI);
        else DIDKM(TL(LI),KI);
        close;
end;

FUNCTION BCHK CHLOC HML KC BC WC=>BL;
VARS CAPTAB CP HLK HLC XK YK XC YC,XX;
        nil->BL;
        If KC="BK" then COPY(WC)->CAPTAB;[WR WN]->CP;
        else [BR BN]->CP;COPY(BC)->CAPTAB;
        close;
LOOP:   If HML.NULL then CHKPC(CAPTAB,BL)->BL;EXIT;
        dest(HML)->HML->HLK;
        HLK.TL.HD->XK;CHLOC.HD.HD->XC;HLK.TL.TL.HD->YK;CHLOC.HD.TL.HD->YC;
        INTOF((YK+YC)/2)->XX;
        INTOF((XK+XC)/2)->YY;
        If XK=XC AND ABBS(YK-YC)=1 then [%CP.HD,XC,YC,XK,YK%]::BL->BL;
        elseif XK=XC and BOARD(XK,XX)="BLANK" then [%CP.HD,XC,YC,XK,YK%]::BL->BL;
        elseif YK=YC and BOARD(YY,YC)="BLANK" then [%CP.HD,XC,YC,XK,YK%]::BL->BL;
        close;
        GENKN(XK,YK,XC,YC,CP,BL)->BL;
        If NOT(NULL(CHLOC.TL)) then
        CHLOC.TL.HD.HD->XC;CHLOC.TL.HD.TL.HD->YC;
        GENKN(XK,YK,XC,YC,CP,BL)->BL;
        close;
        goto LOOP;
end;

FUNCTION RCHK CHLOC HML KC BCAP WCAP => RL;
VARS HPL CP HLK XK YK XC YC CAPTAB;
NIL -> RL; HML -> HPL;
IF KC="BK" THEN WCAP -> CAPTAB; [WP WN WB] -> CP;
ELSE BCAP -> CAPTAB; [BP BN BB] -> CP; CLOSE;
LOOP: IF HPL.NULL THEN CHKPC(CAPTAB,RL) -> RL; EXIT;
DEST (HPL) -> HPL -> HLK;
HLK.TL.HD -> XK; HLK.TL.TL.HD -> YK; CHLOC.HD.HD -> XC; CHLOC.HD.TL.HD -> YC;
IF CAP="Y" AND CP.HD="BP" AND ABBS(XC-XK)=1 AND (YC-YK)=1
THEN [%CP.HD,XC,YC,XK,YK%]::RL->RL;
ELSEIF CAP="Y" AND CP.HD="WP" AND ABBS(XK-XC)=1 AND (YK-YC)=1
THEN [%CP.HD,XC,YC,XK,YK%]::RL->RL;CLOSE;
IF ABBS(XC-XK)=1 AND ABBS(YC-YK)=1
THEN [%CP.TL.TL.HD,XC,YC,XK,YK%]::RL->RL;CLOSE;
GENKN(XK,YK,XC,YC,CP,RL) -> RL;
IF NOT(NULL(CHLOC.TL)) THEN
CHLOC.TL.HD.HD->XC;CHLOC.TL.HD.TL.HD->YC;
GENKN(XK,YK,XC,YC,CP,RL)->RL;CLOSE;
GOTO LOOP;
END;
FUNCTION QCHK CHL HML => QL;
IF HML.HD.TL.TL.TL.HD=CKL.HD.TL.TL.HD
THEN RCHK(CHL,HML,KC,BCAP,WCAP) -> QL;
ELSE BCHK(CHL,HML,KC,BCAP,WCAP) -> QL;
CLOSE;
END;
FUNCTION MOVLIN KC CHLIN CHI=>ML;
COMMENT could a non-checking piece have moved from the
```

```
                checking line to discover check;
vars    XL HL CP CPC BI KN PN MLS;

        If KC="BK" then WLIST->XL;"WQ"->QU;"WB"->BI;"WN"->KN;"WP"->PN;
        else BLIST->XL;"BQ"->QU;"BB"->BI;"BN"->KN;"BP"->PN;
        close;
comment checking piece in CP;
        CHI.HD->HL;HL.TL.HD->CP;HL.TL->CPC;
comment if CP is knight then return;
        If CP=KN then nil->ML;exit;
        PCMOV(XL)->ML;If ML.null then exit;

comment eliminate queen;
        ELIMCP(QU,ML)->ML;
comment eliminate checking piece;
        ELIMPC(CPC,ML)->ML;
comment If CP is bishop then eliminate other bishop (if any);
        If CP=BI then ELIMCP(BI,ML)->ML;
        close;
comment eliminate moves not involving check line squares;
        ELCHL(ML,CHLIN)->ML;
comment Ilegality check if no caps and king not next to king;
        If CAP="N" then NOCAP(ML,PN)->ML;close;NOK(ML,KC)->ML;
end;


FUNCTION ELIMCP PC ML;
        IF ML.null then nil;
        elseif ML.HD.HD=PC then ELIMCP(PC,TL(ML));
        else ML.HD::ELIMCP(PC,TL(ML));
        close;
end;

FUNCTION ELIMPC CPC ML=>XL;
comment eliminate checking pieces from list;
vars    HL HHL;nil->XL;
loop:   If ML.null then return;
        else dest(ML)->ML->HL;[%HL.HD,HL.TL.HD,HL.TL.TL.HD%]->HHL;
                If not(equal(CPC,HHL)) then HL::XL->XL;close;
        goto loop;
        close;
end;

FUNCTION NOCAP ML PN;
COMMENT if CAP="N" then delete pawn revers move vaptures;
        If ML.null then nil;
        elseif ML.HD.HD=PN and ML.HD.TL.HD/=ML.HD.TL.TL.HD then NOCAP(TL(ML),PN);
        else ML.HD::NOCAP(TL(ML),PN);
        close;
end;

FUNCTION NOK  ML  KC=>YL;
comment check king not next to king;
vars    FL XK YK MLH KI;
        nil->YL;FKING(BLIST,WLIST)->FL;
        If KC="WK" then FL.TL.HD.HD->XK;FL.TL.HD.TL.HD->YK;"BK"->KI;
        else FL.HD.HD->XK;FL.HD.TL.HD->YK;"WK"->KI;
        close;
loop:   If  ML.null then return;
        else dest(ML)->ML->MLH;
        If MLH.HD=KI and ABBS(XK-MLH.TL.TL.TL.HD)>1
                        and ABBS(YK-MLH.TL.TL.TL.TL.HD)>1
                then MLH::YL->YL;
        elseif MLH.HD/=KI then MLH::YL->YL;
```

```
        close;
        goto loop;
        close;
end;


FUNCTION ELCHL ML CL => SL;
VARS HL XL KL SL; CL -> XL;NIL->SL;
LOOP: IF XL.NULL THEN EXIT;
DEST(XL) -> XL -> HL;
DELCL(HL,ML) -> KL;IF NOT(KL.NULL) THEN KL<>SL->SL;CLOSE;
 GOTO LOOP;
END;
FUNCTION DELCL HL ML;
IF ML.NULL THEN NIL;
ELSEIF HL.HD=ML.HD.TL.TL.TL.HD AND HL.TL.HD=ML.HD.TL.TL.TL.TL.HD
THEN ML.HD::DELCL(HL,TL(ML));
ELSE DELCL(HL,TL(ML));CLOSE;
END;
FUNCTION GENKN XK YK XC YC CP BL=>BL;
IF ABBS(XK-XC)=2 AND ABBS(YK-YC)=1 THEN [%CP.TL.HD,XK,YK,XC,YC%]::BL -> BL;
ELSEIF ABBS(XK-XC)=1 AND ABBS(YK-YC)=2 THEN [%CP.TL.HD,XK,YK,XC,YC%]::BL -> BL;
END;
FUNCTION CHKPC CP LI;
VARS TF;
IF LI.NULL THEN NIL;
ELSE KDEL(LI.HD.HD,CP)->TF;
IF TF=TRUE THEN LI.HD::CHKPC(CP,TL(LI));
ELSE CHKPC(CP,TL(LI));CLOSE;CLOSE;
END;
FUNCTION KDEL PC CP;
IF CP.NULL THEN FALSE;
ELSEIF CP.HD=PC THEN TRUE;
ELSE KDEL(PC,TL(CP));CLOSE;
END;

FUNCTION PROM CHLIN KC=>PL;
VARS PPL YP PWN NP XL HL XC XH;
NIL->PPL;
IF KC="WK" THEN 2->YP;"BP"->PWN;1->YH; NPB->NP;
ELSEIF KC="BK" THEN 7->YP;"WP"->PWN;8->YH;NPW->NP;
ELSE NIL->PL;EXIT;FNSQ(CHLIN,YP)->XL;
IF NP=8 THEN NIL->PL;EXIT;FNSQ(CHLIN,YP)->XL;
LOOP:IF XL.NULL THEN SHUFFLE(PPL)->PPL;LPC(PPL)->PL;EXIT;
DEST(XL)->XL->HL;HL.HD->XC;
IF CAP="N" THEN [%PWN,XC,YP,XC,YH%]::PPL->PPL;
ELSE [%[%PWN,XC,YP,XC-1,YH%],[%PWN,XC,YP,XC,YH%],[%PWN,XC,YP,XC+1,YH%]%]<>PPL->PPL;
CLOSE;GOTO LOOP;
END;
FUNCTION SHUFFLE XL=>ML;
VARS ANS;TAKE(XL)->XL;NIL->ML;
LOOP:IF XL.NULL THEN EXIT;
MEMLST(XL.HD,TL(XL))->ANS;IF ANS=TRUE THEN XL.HD::ML->ML;CLOSE;
TL(XL)->XL;GOTO LOOP;
END;
FUNCTION TAKE XL=>YL;
VARS X Y HL;NIL->YL;
LOOP:IF XL.NULL THEN RETURN;
ELSE DEST(XL)->XL->HL;HL.TL.TL.TL.HD->X;HL.TL.TL.TL.TL.HD->Y;
IF X>8 OR X<1 OR BOARD(X,Y)=KC THEN GOTO LOOP;
ELSE HL::YL->YL;GOTO LOOP;CLOSE;CLOSE;
END;
FUNCTION LPC PPL=>ML;
VARS HL HHL XL LH YL;NIL->ML;
```

```
LOOP:IF PPL.NULL THEN EXIT;
PPL.HD->HL;[%HL.TL.TL.TL.HD,HL.TL.TL.TL.TL.HD%]->HHL;
IF KC="WK" THEN BLIST->XL;ELSE WLIST->XL;CLOSE;
LOOP1:IF XL.NULL THEN TL(PPL)->PPL;GOTO LOOP;
ELSE [%XL.HD.TL.HD,XL.HD.TL.TL.HD%]->LH;
IF EQUAL(HHL,LH) THEN HL::ML->ML;CLOSE;CLOSE;
TL(XL)->XL;GOTO LOOP1;
END;
FUNCTION FNSQ CL PPR;
IF CL.NULL THEN NIL;
ELSEIF CL.HD.TL.HD=PPR THEN CL.HD::FNSQ(TL(CL),PPR);
ELSE FNSQ(TL(CL),PPR);CLOSE;
END;

FUNCTION COUNTB CPT CB;
COMMENT          count the number of bishops on board;
        If CPT.null then CB;
        elseif CPT.HD="WB" or CPT.HD="BB" then CB+1->CB;COUNTB(TL(CPT),CB);
        else COUNTB(TL(CPT),CB);
        close;
end;
FUNCTION WHPCAP HL=>CL;
COMMENT HL=[PC X Y] what was captured by pc on (x,y)
                if not underpromotions us bleft/wleft;

vars    PC XP YP BW CPT HT CPH CB ANS LIS2 LIS3 CB CPTX FLG;
COMMENT  if a piece is not a checking piece set FLG=1,else FLG=0;
        HL.HD->PC;HL.TL.HD->XP;HL.TL.TL.HD->YP;BORW(PC)->BW;nil->LIS2;nil->LIS3;

        If BW="B" and UPFLG=0 THEN WLEFT->CPT;WHTAB->HT;
        elseif BW="B" then [WP WN WR WQ WB]->CPT;WHTAB->HT;
        elseif BW="W" and UPFLG=0 then BLEFT->CPT;BHTAB->HT;
        elseif BW="W" then [BP BN BR BQ BB]->CPT;BHTAB->HT;
        close;
        0->CB;0->FLG;COUNTB(CPT,CB)->CB;
COMMENT form short list;

        SRTLFT(CPT)->CPT; COPYALL(CPT)->CPTX;

loop:   If CPT.null then [%CPTX,LIS2,LIS3%]->CL;goto L1;
        else dest(CPT)->CPT->CPH;
                If CPH="BP" or CPH="WP" then PCH(XP,YP,CPH,HT);
                elseif CPH="BB" or CPH="WB" then BCH(CB,CPH,XP,YP);
                else GCH(XP,YP,CPH);
                close;

                goto loop;
        close;
L1:     If lengt(CPTX)=LENGT(LIS2) and FLG=0 then 1->ILLFLG;close;
end;

FUNCTION GCH X Y P;
vars    MLX RL PC RLH ML ANS;
        COPYALL(KL)->RL;nil->ML;BORW(HL.HD)->PC;COPYALL(MLIST)->MLX;
        0->LEGFLG;

loop:   If RL.null then COPYALL(MLX)->MLIST;[%P%]<>LIS2->LIS2;return;
        else dest(RL)->RL->RLH;
                If RLH.HD/=HL.HD then [%nil,RLH%]::LIS3->LIS3;1->FLG;goto loop;
                elseif PC="W" then UPDATE(RLH,WLIST);
                else UPDATE(RLH,BLIST);
                close;

        PCMOV([%[%P,X,Y%]%])->MLIST;
```

```
COMMENT subset of legality checker applied;
        .RULE14->ANS;.RULE16->ANS;.RULE17->ANS;.RULE18->ANS;RESTOR(RLH);
        COPYALL(MLIST)->ML;
        If not(ML.null) then COPYALL(MLX)->MLIST;RLH::ML->ML;ML::LIS3->LIS3;return;
                else goto loop;
                close;
        close;
end;

FUNCTION PCH XP YP CPH HT;
vars    X RNK;
        If CPH="BP" then 7->RNK;
        else 2->RNK;
        close;
        GABS(XP,HT)->X;
        If ABBS(YP-RNK)<ABBS(X-XP) then [%CPH%]<>LIS2->LIS2;
        else  GCH(XP,YP,CPH);
        close;
end;

FUNCTION BCH CB CPH XP YP;
vars XL BSQ XB YB;
        If CB=2 then  GCH(XP,YP,CPH);return;
        elseif CPH="BB" then BLIST->XL;
        else WLIST->XL;
        close;
        LFORB(XL)->BSQ;

        If not(BSQ.null) then BSQ.HD->XB;BSQ.TL.HD->YB;
                If CLSQ(XP,YP,XB,YB)  or LCONSQ(CPH,XP,YP) then
                        GCH(XP,YP,CPH);
                else [%CPH%]<>LIS2->LIS2;
                close;
        close;
end;

FUNCTION LFORB XL;
COMMENT look for bishop;
        If XL.null then nil;
        elseif XL.HD.HD="BB" or XL.HD.HD="WB"
                then [%XL.HD.TL.HD,XL.HD.TL.TL.HD%];
        else LFORB(TL(XL));
        close;
end;

FUNCTION LCONSQ CPH XP YP=>REP;
vars    XB YB;
If CPH="BB" and BOARD(2,7)="BP" and BOARD(4,7)="BP" then 3->XB;8->YB;
elseif CPH="BB" and BOARD(5,7)="BP" and BOARD(7,7)="BP" then 6->XB;8->YB;
elseif CPH="WB" and BOARD(2,2)="WP" and BOARD(4,2)="WP" then 3->XB;1->YB;
elseif CPH="WB" and BOARD(5,2)="WP" and BOARD(7,2)="WP" then 6->XB;1->YB;
close;
CLSQ(XP,YP,XB,YB)->REP;
end;

FUNCTION CLSQ XP YP XB YB;
If ERASE((XB+YB)//2)/=ERASE((XP+YP)//2) then true;else false;close;
end;

FUNCTION SRTLFT XL=>YL;
COMMENT form short list from xl.delete repetitions except for bishops;

vars    HL;nil->YL;
loop:   If XL.null then return;
```

```
            else dest(XL)->XL->HL;HL::YL->YL;
                    If HL/="BB" and HL/="WB" then DELT(HL,XL)->XL;
                    close;
            goto loop;
            close;
    end;
```

```
LLL             EEEEEEEEEEEEEEEE        GGGGGGGGGGGG        RRRRRRRRRRRR
LLL             EEEEEEEEEEEEEEEE        GGGGGGGGGGGG        RRRRRRRRRRRR
LLL             EEEEEEEEEEEEEEEE        GGGGGGGGGGGG        RRRRRRRRRRRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEE                   GGG                   RRR         RRR
LLL             EEEEEEEEEEEEE         GGG                   RRRRRRRRRRRR
LLL             EEEEEEEEEEEEE         GGG                   RRRRRRRRRRRR
LLL             EEEEEEEEEEEEE         GGG                   RRRRRRRRRRRR
LLL             EEE                   GGG   GGGGGGGGG       RRR   RRR
LLL             EEE                   GGG   GGGGGGGGG       RRR   RRR
LLL             EEE                   GGG   GGGGGGGGG       RRR   RRR
LLL             EEE                   GGG         GGG       RRR     RRR
LLL             EEE                   GGG         GGG       RRR     RRR
LLL             EEE                   GGG         GGG       RRR     RRR
LLLLLLLLLLLLLLL EEEEEEEEEEEEEEEE        GGGGGGGGG           RRR       RRR
LLLLLLLLLLLLLLL EEEEEEEEEEEEEEEE        GGGGGGGGG           RRR       RRR
LLLLLLLLLLLLLLL EEEEEEEEEEEEEEEE        GGGGGGGGG           RRR       RRR

PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPP         PPP       OOO         OOO   PPP         PPP
PPP         PPP       OOO         OOO   PPP         PPP
PPP         PPP       OOO         OOO   PPP         PPP
PPP         PPP       OOO         OOO   PPP         PPP
PPP         PPP       OOO         OOO   PPP         PPP
PPPPPPPPPPPP          OOO         OOO   PPPPPPPPPPPP
PPPPPPPPPPPP          OOO         OOO   PPPPPPPPPPPP
PPPPPPPPPPPP          OOO         OOO   PPPPPPPPPPPP
PPP                   OOO         OOO   PPP
PPP                   OOO         OOO   PPP
PPP                   OOO         OOO   PPP
PPP                   OOO         OOO   PPP                     ......
PPP                   OOO         OOO   PPP                     ......
PPP                   OOO         OOO   PPP                     ......
PPP                     OOOOOOOOO       PPP                     ......
PPP                     OOOOOOOOO       PPP                     ......
PPP                     OOOOOOOOO       PPP                     ......

*START* Job BAGAT Req #751 for O.ENABLEDOP    Date 11-Jun-86 10:53:58 Monitor: O
File RS:<R.ALDEN>LEGRUL.POP.1, created: 13-Apr-84 12:46:35, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:56   Page limit:225   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION LEGCHK CAP XL=>LLIST;
COMMENT reverse move legality checker;
vars ALR AH ITM HL ANS DDBFX;nil->DDBFX;
comment check if move list empty and whether explanation required
                LEGFLG set for explanation;
        If XL.null then nil->LLIST; exit;
        COPYALL(ALLRULES)->ALR;COPYALL(XL)->MLIST;
COMMENT go through each legality rule in turn;
loop:   If not(ALR.null) then dest(ALR)->ALR->AH;
                If category(valof(AH))="LEGRULE"
                then eval(premise(valof(AH)))->ANS;
                close;
        goto loop;close;

MLIST->LLIST;
COMMENT if no illegal moves then file null in LEGDEL
                providing LEGFLG=1;
        If LEGFLG=0 then return;
        elseif not(LLIST.null) then [%[LIST],LLIST,[R41EX]%]::DDBFX->DDBFX;
        close;
        If not(DDBFX.null) and LEGFLG=1 then DDBFX<>DDB->DDB;close;
        0->LEGFLG;
        end;
FUNCTION DSET HL AH=>DDBFX;
COMMENT place entries from rules in data base;
        [%[%[LEGRUL],HL,explain(valof(AH))%]%]<>DDBFX->DDBFX;
end;


FUNCTION RULE18=>ANS;
COMMENT no captures permitted - delete all reverse pawn move captures;

vars    ML MLH ML;
        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
loop:   If not(CAP="N") then return;
        elseif ML.null and LEGDEL.null then return;
        elseif ML.null  and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->MLH;MLH.HD->MLHD;
                If MLHD/="WP" and MLHD/="BP" then goto loop;
                elseif MLH.TL.HD/=MLH.TL.TL.TL.HD
                then DELTL(MLH,MLIST)->MLIST;MLH::LEGDEL->LEGDEL;
                        true->ANS;
                close;
        goto loop;
        close;
end;

FUNCTION RULE13=>ANS;
COMMENT legality check.king not adjacent to king;
vars    FL P XK YK ML MLH;

COMMENT find  present king co-ords,copy MLIST(move list)
                reset LEGDEL;

        FKING(BLIST,WLIST)->FL;nil->LEGDEL;COPYALL(MLIST)->ML;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="W" then FL.TL.HD.HD->XK;FL.TL.HD.TL.HD->YK;
                else FL.HD.HD->XK;FL.HD.TL.HD->YK;
                close;
        close;
COMMENT now check if kings adjacent;
```

```
loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->MLH;
                If MLH.HD/="BK" and MLH.HD/="WK" then goto loop;
                elseif not(ABBS(XK-MLH.TL.TL.TL.HD)>1)
                and not(ABBS(YK-MLH.TL.TL.TL.TL.HD)>1)
                then DELTL(MLH,MLIST)->MLIST;MLH::LEGDEL->LEGDEL;true->ANS;
                close;
        goto loop;
        close;
end;

FUNCTION RULE14=>ANS;
COMMENT legality check.opposition king in check;
vars    ML P KI HL CHI KC KL;

COMMENT set up variables;

        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="B" then "WK"->KI;
                else "BK"->KI;
                close;
        close;

COMMENT         now check for oppo king;
loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;
                If P="B" then UPDATE(HL,BLIST);else UPDATE(HL,WLIST);
                close;
        close;
        ISKCHK(BLIST,WLIST)->CHI->KC;
                If CHI=false then goto L2;
                elseif CHI.null then goto L1;
                elseif KC=KI
                and equal([%CHI.HD.TL.TL.HD,CHI.HD.TL.TL.TL.HD%],[%HL.TL.TL.TL.HD,HL.TL.
                        then goto L2;                                   TL.TL.TL.TL.HD%])
                else goto L1;
                close;
L2:     DELTL(HL,MLIST)->MLIST;true->ANS;HL::LEGDEL->LEGDEL;
L1:     RESTOR(HL);goto loop;
end;

FUNCTION RULE15=>ANS;
COMMENT legality check.own king in check and no way found for this
                to happen;
vars    ML P KI HL CHI KC KL;

COMMENT set up variables;

        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="B" then "BK"->KI;
                else "WK"->KI;
                close;
        close;

COMMENT see if own king in check;
```

```
loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;
                If P="B" then UPDATE(HL,BLIST);else UPDATE(HL,WLIST);
                close;
        close;
        ISKCHK(BLIST,WLIST)->CHI->KC;
                If CHI=false then goto L2;
                elseif CHI.null then goto L1;
                elseif KC=KI and not(CHPWN(CHI)) then goto L2;
                elseif KC=KI then KCHECK1(CHI,KC)->KL;
                        If not(KL.null) then goto L1;close;
                close;

L2:     DELTL(HL,MLIST)->MLIST;true->ANS;HL::LEGDEL->LEGDEL;
L1:     RESTOR(HL);goto loop;
end;

FUNCTION RULE16=>ANS;
COMMENT legality check.too many pawn captures;
vars    ML P BTO TOT BN WN WH BH PC;

COMMENT set up variables;
        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="B" then WTOT->BTO;
                else BTOT->BTO;
                close;
        close;

COMMENT check possible pawn captures;

loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;HL.HD->PC;
                If PC="BP" then UPDATE(HL,BLIST);
                elseif PC="WP" then UPDATE(HL,WLIST);
                else goto loop;
                close;
        .MINPWN->WH->BH->WN->BN;
        If PC="BP" then BN.HD->TOT;else WN.HD->TOT;
        close;
comment tot contains total pawn captures;
                        If HL.TL.HD/=HL.TL.TL.TL.HD and TOT+1>BTO
                                then true->ANS;HL::LEGDEL->LEGDEL;
                                DELTL(HL,MLIST)->MLIST;
                                close;
        RESTOR(HL);goto loop;
        close;
end;

FUNCTION RULE17=>ANS;
COMMENT legality check.if no.of pieces captured on board
                is less than no.of opposition pawn captures;

vars    ML P BX HL WL BL TC;
COMMENT set up variables;
        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="B" then WPTT->BX;else BPTT->BX;
```

```
                close;
        close;

COMMENT check pawn captures;
loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;
                If HL.HD="BP" then UPDATE(HL,BLIST);
                elseif HL.HD="WP" then UPDATE(HL,WLIST);
                else goto loop;
                close;
                .CPGTOB->WL->BL;

                        If  HL.HD="BP" then LENGT(BL)->TC;
                        else LENGT(WL)->TC;
                        close;
                        If TC<BX then true->ANS;HL::LEGDEL->LEGDEL;
                        DELTL(HL,MLIST)->MLIST;
                        close;
        RESTOR(HL);goto loop;
        close;
end;

FUNCTION RULE12=>ANS;
COMMENT legality check.piece given as not moving(PNM);
vars    ML HL;
        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;
                If not(MEMLST([%HL.HD,HL.TL.HD,HL.TL.TL.HD%],PNM))
                then true->ANS;HL::LEGDEL->LEGDEL;DELTL(HL,MLIST)->MLIST;
                close;
        goto loop;
        close;
end;

FUNCTION RULE19=>ANS;
COMMENT legality check.a pawn reverse moves to a home square and
                constrains a piece that was captured on board;
vars    ML HL RNK WL BL YL P TOT;

COMMENT set up variables;

        COPYALL(MLIST)->ML;nil->LEGDEL;false->ANS;
        If ML.null then return;
        else BORW(ML.HD.HD)->P;
                If P="B" then BTOT->TOT;
                else WTOT->TOT;
                close;
        close;

COMMENT check pawns;

loop:   If ML.null and LEGDEL.null then return;
        elseif ML.null and LEGFLG=0 then return;
        elseif ML.null then DSET(LEGDEL,AH)->DDBFX;return;
        else dest(ML)->ML->HL;HL.TL.TL.TL.TL.HD->RNK;
                If HL.HD="WP" and RNK=2 then UPDATE(HL,WLIST);
                elseif HL.HD="BP" and RNK=7 then UPDATE(HL,BLIST);
                else goto loop;
                close;
```

```
                .CPGTOB->WL->BL;
                        If HL.HD="WP" then lengt(WL)->YL;
                        else lengt(BL)->YL;
                        close;
                        If YL/=TOT then true->ANS;HL::LEGDEL->LEGDEL;
                        DELTL(HL,MLIST)->MLIST;
                        close;
        RESTOR(HL);goto loop;
        close;

end;

FUNCTION RESTOR XL;
COMMENT to restore a given move;
vars BBW X1 Y1 X2 Y2;BORW(XL.HD)->BBW;
XL.TL.HD->X1;XL.TL.TL.HD->Y1;XL.TL.TL.TL.HD->X2;XL.TL.TL.TL.TL.HD->Y2;
        If EQUAL([%X1,Y1%],[%X2,Y2%]) then exit;
        "BLANK"->BOARD(X2,Y2);
If SFLAG=1 AND BBW="B" then TL(BLIST)->BLIST;"BLANK"->BOARD(X1,Y1);0->SFLAG;return;
elseif SFLAG=1 AND BBW="W" then TL(WLIST)->WLIST;"BLANK"->BOARD(X1,Y1);0->SFLAG;
close;
XL.HD->BOARD(X1,Y1);
        If BBW="B" then RSTOR(XL,BLIST);else RSTOR(XL,WLIST);
        close;
        0 ->SFLAG;
end;

FUNCTION RSTOR XL LI;
        If XL.TL.TL.TL.HD=LI.HD.TL.HD and XL.TL.TL.TL.TL.HD=LI.HD.TL.TL.HD
        then XL.TL.HD->LI.HD.TL.HD;XL.TL.TL.HD->LI.HD.TL.TL.HD;
        else RSTOR(XL,TL(LI));
        close;
end;

FUNCTION UPDATE XL LIS;
COMMENT to update board and list of pieces for given move;
VARS PC LIST X1 Y1 X2 Y2;0->SFLAG;
XL.TL.HD->X1;XL.TL.TL.HD->Y1;XL.TL.TL.TL.HD->X2;XL.TL.TL.TL.TL.HD->Y2;
        If EQUAL([%X1,Y1%],[%X2,Y2%]) then exit;
        XL.HD->BOARD(X2,Y2);"BLANK"->BOARD(X1,Y1);RPLACE(XL,LIS);
        If SFLAG=0 then return;
        else [%XL.HD,X2,Y2%]->LIST;BORW(XL.HD)->PC;
                If PC="B" then LIST::BLIST->BLIST;
                else LIST::WLIST->WLIST;
                close;
        close;
end;
FUNCTION RPLACE XL LI;
        If LI.null then 1->SFLAG;
        elseif XL.TL.HD=LI.HD.TL.HD and XL.TL.TL.HD=LI.HD.TL.TL.HD
                then XL.TL.TL.TL.HD->LI.HD.TL.HD;
                        XL.TL.TL.TL.TL.HD->LI.HD.TL.TL.HD;
        else RPLACE(XL,TL(LI));
        close;
end;



FUNCTION LPN J Y Y1 Y2 LST=>LST;
COMMENT look up column for pawns;
        If BOARD(J,Y1)=PWN then [%J,Y%]::LST->LST;
        elseif BOARD(J,Y2)=PWN then [%J,Y1%]::LST->LST;[%J,Y%]::LST->LST;
        else 0->FLAG;
```

```
        close;
end;

FUNCTION LCOL X Y PWN=>LST;
vars FLAG;NIL->LST;1->FLAG;
        If Y=9 then 8->Y;elseif Y=0 then 1->Y;close;
        If X=1 and PWN="WP" then FORALL J X 1 Y LPN(J,1,2,3,LST)->LST;close;
        Elseif X=8 and PWN="WP" then FORALL J Y 1 X LPN(J,1,2,3,LST)->LST;close;
        Elseif X=1 and PWN="BP" then FORALL J X 1 Y LPN(J,8,7,6,LST)->LST;close;
        Elseif X=8 and PWN="BP" then FORALL J Y 1 X LPN(J,8,7,6,LST)->LST;close;
        Elseif Y=8 and PWN="WP" then FORALL J X 1 Y LPN(J,1,2,3,LST)->LST;close;
        Elseif Y=8 and PWN="BP" then FORALL J X 1 Y LPN(J,8,7,6,LST)->LST;close;
        close;
        If FLAG=0 then NIL->LST;close;
end;
FUNCTION LBROW PWN XXL=>ARU ARC;
COMMENT look along each row;
VARS NC ADD CL BL P X Y XX PX PY;NIL->ARU;
XXL.HD->P;XXL.TL.HD->X;XXL.TL.TL.HD->Y;
        If X=1 then 1->ADD;-1->PX;1->NC;else -1->ADD;1->PX;8->NC;close;
        If PWN="WP" then 1->PY;else -1->PY;close;
LOOP:IF NC=9 or NC=0 then LCOL(1,8,PWN)->ARC;return;
        Elseif BOARD(NC,Y)=P then [%P,NC,Y%]->HP;close;
        If BOARD(NC,Y)="BLANK" or BOARD (NC,Y)=P then NC+ADD->NC;goto loop;
        else [%BOARD(NC,Y),NC,Y%]->BL;
        If ADD=1 then NC-1->XX;LCOL(X,XX,PWN)->ARC;
        else NC+1->XX;LCOL(XX,X,PWN)->ARC;
        close;close;
        If ARC.NULL then EXIT;MEMLST(BL,PNM)->CL;
        If CL=TRUE and BOARD(NC,Y+PY)=PWN then BL::ARU->ARU;
        elseif CL=TRUE and BOARD(NC+PX,Y+PY)=PWN then BL::ARU->ARU;
        elseif CL=FALSE and BOARD(NC,Y+PY)=PWN then NIL->ARU;return;
        elseif CL=FALSE and BOARD(NC+PX,Y+PY)=PWN then NIL->ARU;return;
        elseif CL=FALSE and BOARD(NC,Y+PY)/="BLANK" then [%BOARD(NC,Y+PY),NC,PY%]->BL;
                If MEMLST(BL,PNM)=false then NIL->ARU;return;else BL::ARU->ARU;
                close;
        close;
        If ADD=1 then LCOL(1,NC,PWN)->ARC;else LCOL(NC,8,PWN)->ARC;close;
        If ARC.NULL then return;
        else NC+ADD->NC;goto loop;
        close;
end;
FUNCTION MEMLST XL1 XL2;
COMMENT  if xl1 is a member of xl2 then false - else true;
        If XL2.NULL then true;
        elseif equal(XL1,XL2.HD) then false;
        else MEMLST(XL1,TL(XL2));
        close;
end;

FUNCTION RBRD XXL=>YRC YRU;
COMMENT look for rook constraints;
        If XXL.HD="WR" then LBROW("WP",XXL)->YRC->YRU;
        else LBROW("BP",XXL)->YRC->YRU;
        close;
end;
FUNCTION CPGTOB=>BL WL;
COMMENT returns list of pieces captured on board;
vars    RC RU BC QC;
        GENCON("W")->QC->BC->RU->RC;GENCON("B")->QC->BC->RU->RC;
        BLEFT->BL;WLEFT->WL;
end;

FUNCTION GENCON P=>RC RU BC QC;
```

```
vars    WH BH XL N BI X Q CB CBH HL K KC CONSQ R1 R2 RU1 RU2 RC1 RC2 PWN RN;
vars    NBLIST NWLIST PNMX I;

comment TO RETURN A LIST OF PIECES THAT WERE CAPTURED ON BOARD;
        COPYALL(WLIST)->NWLIST;COPYALL(BLIST)->NBLIST;COPYALL(PNM)->PNMX;

        nil->QC;nil->BC;nil->RC;nil->RU;nil->CONSQ;
comment GET LISTS OF BISHOPS CAPTURED AT HOME IN WH,BH;
        .BCHSQ->WH->BH;

        If P="W" then WHCAP->XL;"WP"->PWN;1->RN;"WN"->N;"WB"->BI;WH->X;"WQ"->Q;
                [WQ 4 1]->HL;[WR 1 1]->R1;[WR 8 1]->R2;"WK"->K;[WK 5 1]->KL;
        else BLCAP->XL;"BP"->PN;8->RN;"BN"->N;"BB"->BI;BH->X;"BQ"->Q;[BQ 4 8]->HL;
                "BK"->K;[BR 1 8]->R1;[BR 8 8]->R2;[BK 5 8]->KL;
        close;

COMMENT remove any knights from back row;

        FORALL I 1 1 8 IF BOARD(I,RN)=N then "BLANK"->BOARD(I,RN);close;close;
comment DELETE BISHOPS CAPTURED AT HOME;
        If lengt(X)=1 then Dl(BI,XL)->XL;
        elseif lengt(X)=2 then DELT(BI,XL)->XL;
        close;
comment GET LIST OF BISHOPS CONSTRAINED TO HOME SQUARE;
        LCONS1(BI)->CB;
loop:   If not(CB.null) then dest(CB)->CB->CBH;CBH::PNM->PNM;
                [%CBH,CBH.TL%]::BC->BC;goto loop;
        close;

comment NOW LOOK AT QUEEN;
        If membl(HL,PNM) then [%HL,HL.TL%]::QC->QC;
        elseif ISBR(Q) and CONST(Q) then HL::PNM->PNM;[%HL%]<>CONSQ::QC->QC;
        elseif not(ISBR(Q)) and memb(Q,XL) and CONST(Q) then DELT(Q,XL)->XL;
        close;

comment NOW FOR KING;
        If not(membl(KL,PNM)) and ISBR(K) and not(CONSQ.null)
                then KL::PNM->PNM;
        close;


COMMENT         look at rook constraints,returned is lists of those
                rooks constrained and those that coul escape if certain
                pieces moved;

        RBRD(R1)->RU1->RC1;RBRD(R2)->RU2->RC2;
        PPNM(RU1)->RU1;PPNM(RU2)->RU2;
        If not(RU1.null) then R1::RU1->RU1;RU1::RU->RU;close;
        If not(RU2.null) then R2::RU2->RU2;RU2::RU->RU;close;
        If not(RC1.null) then R1::RC1->RC1;RC1::RC->RC;close;
        If not(RC2.null) then R2::RC2->RC2;RC2::RC->RC;close;

        If lengt(RC)=1 then Dl(R1.HD,XL)->XL;
        elseif lengt(RC)=2 then DELT(R1.HD,XL)->XL;
        close;

        If P="W" then XL->WLEFT;else XL->BLEFT;
        close;
COMMENT restore original board position;
        COPYALL(NWLIST)->WLIST;COPYALL(NBLIST)->BLIST;COPYALL(PNMX)->PNM;.RSTB;
end;

FUNCTION PPNM XL=>YL;
```

```
vars    XLH;nil->YL;
COMMENT xl is a list such that a rook is unconstrained if csertain
                pieces can move.if theses pieces are in PNM  then they
                cannot escape from the back row,so delete;

loop:   If XL.null then return;
        else dest(XL)->XL->XLH;
                If not(membl(XLH,PNM)) then XLH::YL->YL;
                close;
        goto loop;
        close;
end;

FUNCTION ISBR P;
vars    XL PC Y;
comment If P is on the back row then true,else false;
BORW(P)->PC;
        If PC="B" then 8->Y;BLIST->XL;
        else 1->Y;WLIST->XL;
        close;
loop:   If XL.null then false;
        elseif XL.HD.HD=P and Y=XL.HD.TL.TL.HD then true;
        else TL(XL)->XL;goto loop;
        close;
end;

FUNCTION CONST Q;
vars RNK BL BL1 XL KL PN BRNK;
comment IF QUEEN CONSTRAINED RETURN TRUE AND LIST OF CONSTRAINING
                SQUARES IN CONSQ;
        BORW(Q)->PC;
        If PC="W" then 2->RNK;[WB 3 1]->BL;WH->XL;[WK 5 1]->KL;"WP"->PN;
                [WB 6 1]->BL1;1->BRNK;
        else 7->RNK;[BB 3 8]->BL;BH->XL;[BK 5 8]->KL;"BP"->PN;
                [BB 6 8]->BL1;8->BRNK;
        close;

        If membl(BL,PNM) and membl(KL,PNM) and FALL(3,5)
        then true;FPL(4,4,CONSQ)->CONSQ;
        elseif membl(KL,PNM) and FALL(1,5) then true;FPL(1,4,CONSQ)->CONSQ;
        elseif membl(BL,PNM) and membl(BL1,PNM) and FALL(3,6)
                then FPL(4,5,CONSQ)->CONSQ;true;
        elseif  membl(BL,PNM) and  FALL(3,8) then FPL(4,8,CONSQ)->CONSQ;true;
        elseif membl(BL1,PNM) and FALL(1,6) then FPL(1,5,CONSQ)->CONSQ;true;
        elseif FALL(1,8) then FPL(1,7,CONSQ)->CONSQ;true;
        else false;
        close;
end;

FUNCTION FALL N1 N2;
COMMENT are ther any pawns at home from n1 to n2;
loop:   If N1>N2 then true;
        elseif BOARD(N1,RNK)/=PN then false;
        else N1+1->N1;goto loop;
        close;
end;

FUNCTION FPL N1 N2 CL=>CL;
COMMENT place squares in c1;
loop: If N1>N2 then return;
        else [%N1,BRNK%]::CL->CL;1+N1->N1;goto loop;
        close;
end;
```

```
FUNCTION LCONS1 B=>BCC;
COMMENT return list of bishops constrained at home;
        nil->BCC;
If B="BB" and BOARD(3,8)="BB" and BOARD(2,7)="BP" and BOARD(4,7)="BP"
        then [BB 3 8]::BCC->BCC;close;
If B="BB" and BOARD(6,8)="BB" and BOARD(5,7)="BP" and BOARD(7,7)="BP"
        then [BB 6 8]::BCC->BCC;close;
If B="WB" and BOARD(3,1)="WB" and BOARD(2,2)="WP" and BOARD(4,2)="WP"
        then [WB 3 1]::BCC->BCC;close;
If B="WB" and BOARD(6,1)="WB" and BOARD(5,2)="WP" and BOARD(7,2)="WP"
        then [WB 6 1]::BCC->BCC;close;
end;
```

```
FUNCTION PARS XL;
COMMENT to parse the given question and set flags;

vars N;
        NIL->MISPQ;0->MISPC;0->WCFLGX;0->BCFLGX;0->BCFLG;0->WCFLG;0->NC;
        nil->MSPC;0->CAPPC;
        If MEMB("LAST",XL) AND MEMB("MOVE",XL) then FNINT(XL)->N;
                If N.ISINTEGER then N->NC;N->CP;close;
        elseif MEMB("WHAT",XL) and MEMB("MISSING",XL)
        then FNLST(XL);1->MSFLG;1->MSFLGX;
        elseif MEMB("WHAT",XL) and MEMB("SQUARE",XL) and MEMB("CAPTURED",XL)
                then FNLST(XL);MISPC->CAPPC;0->MISPC;
        elseif MEMB("LAST",XL) and MEMB("MOVES",XL) then FNINT(XL)->N;
                If N.ISINTEGER THEN N->NC;N->CP;close;
        elseif MEMB("CASTLE",XL) or MEMB("CASTLING",XL) then FNCAS(XL)->N;
                If N="B" then lengt(BCAS)->BCFLG;COPYALL(BCFLG)->BCFLGX;
                elseif N="W" then lengt(WCAS)->WCFLG;COPYALL(WCFLG)->WCFLGX;
                else lengt(BCAS)->BCFLG;lengt(WCAS)->WCFLG;
                        copyall(BCFLG)->BCFLGX;copyall(WCFLG)->WCFLGX;
                close;
        elseif TWOL(XL) then LOCAT(XL);
        elseif memb("PROMOTED",XL) and MEMB("BOARD",XL) then 1->PPONB;
        COPYALL(PPONB)->PPONBX;
        else .NONQ;
        close;
end;


FUNCTION NONQ;
NL(1);prstring('UNABLE TO PARSE QUESTION,BUT DEDUCTIONS WILL BE CARRIED OUT!);
NL(1);
END;
```

```
FUNCTION FNCAS XL;
COMMENT possible castling problem - is it black or white or both;

        If XL.null then "U"; return;
        elseif XL.HD="B" or XL.HD="BLACK" then "B";
        elseif XL.HD="W" or XL.HD="WHITE" then "W";
        else FNCAS(TL(XL));
        close;
end;

FUNCTION MEMB X XL;
COMMENT if x is a member of xl then true else false;
        If XL.NULL then false;
        elseif X=XL.HD then true;
        else MEMB(X,TL(XL));
```

```
        close;
end;

FUNCTION MEMBL X XL;
COMMENT if list x is a member of list xl then true - else false;
        If XL.null then false;
        elseif equal(X,XL.HD) then true;
        else MEMBL(X,TL(XL));
        close;
end;

FUNCTION FNINT XL;
COMMENT look through list xl to see if contains integer;
        If XL.null then "A";
        elseif XL.HD.ISINTEGER then XL.HD;
        else FNINT(TL(XL));
        close;
end;

FUNCTION FNLST XL;
COMMENT missing piece - find if piece or square given
        if square place in MISPQ else MISPQ null
        if piece place in MISPC else MISPC zero;

vars YL;
        XL->YL;
LOOP:   If YL.null then return;
        elseif YL.HD.ISLIST then YL.HD->MISPQ;
        elseif BRW(YL.HD) then YL.HD->MISPC;
        else TL(YL)->YL;goto loop;
        close;
end;

FUNCTION BRW X;
COMMENT find if piece given;
        If X="BP" or X="BB" or X="BN" or X="BR" or X="BK" or X="BQ"
                or X="WP" or X="WB" or X="WN" or X="WK" or X="WQ" or X="WR"
                then true;
        else false;
        close;
end;


FUNCTION TWOL XL;
COMMENT if there two lists in xl then possible location problem;

vars YL YLH;
        nil->PLOC;COPYALL(XL)->YL;
loop:   If YL.NULL then goto L1;
        else dest(YL)->YL->YLH;
                If YLH.ISLIST then YLH::PLOC->PLOC;
                close;
        goto loop;
        close;

L1:     If lengt(PLOC)=2 then true;
        else nil->PLOC;false;
        close;
end;

FUNCTION LOCAT XL;
COMMENT location problem determine piece;
vars    YL XLH H1 H2;
        nil->YL;
```

```
loop:   If XL.null then false;return;
        else dest(XL)->XL->XLH;
                If BRW(XLH) then PLOC.HD->H1;PLOC.TL.HD->H2;
                        XLH::H1->H1;XLH::H2->H2;H2::YL->YL;
                        H1::YL->YL;YL->PLOC;true;1->LOCFLG;6->CP;return;
                close;
        goto loop;
        close;
end;



FUNCTION PARSEC XL;
COMMENT to pars initial conditions;

        If memb("LAST",XL) and memb("MOVE",XL) then LMV(XL);
        elseif memb("LAST",XL) and memb("MOVED",XL) then LMV(XL);
        elseif memb("CAPTURE",XL) or memb("CAPTURES",XL) then CAPM(XL);
        elseif memb("CAN",XL) and memb("CASTLE",XL) then BWCANC(XL);
        elseif memb("COLOUR",XL) and memb("MOVED",XL) then COLM(XL);
        elseif memb("COLOUR",XL) and memb("MOVE",XL) then COLM(XL);
        elseif memb("ODDS",XL) then ODDM(XL);
        elseif memb("ORIGINAL",XL) then ORPM(XL);
        elseif memb("CHECK",XL) then CHM(XL);
        elseif memb("PROMOTED",XL) then 0->PPOB;
        elseif memb("MOVED",XL) and memb("NOT",XL) then MVL(XL);
        elseif memb("UNDERPROMOTIONS",XL) then 0->UPFLG;
        else .NONIC;
        close;
end;

FUNCTION NONIC;
nl(1);prstring('unable to parse initial conditions,so will ignore!);
nl(1);
end;

FUNCTION MVL XL;
COMMENT xl contains pieces that have not moved - locate and place in PNM;

        vars    XXL H P YL;
        copyall(XL)->XXL;

loop:   If not(XXL.null) and BRW(XXL.HD) then XXL.HD->H;
        else TL(XXL)->XXL;goto loop;
        close;

        BORW(H)->P;

        If P="B" then copyall(BLIST)->YL;
        else copyall(WLIST)->YL;
        close;

loopl:  If YL.null then return;
        elseif YL.HD.HD=H then YL.HD::PNM->PNM;
        else TL(YL)->YL;goto loopl;
        close;

end;

FUNCTION BWCANC XL;
COMMENT given that b or w can castle.add to pieces not moved list PNM;

        vars HL;
                If memb("WHITE",XL) or memb("W",XL) then WCAS->HL;
```

```
                else BCAS->HL;
                close;

        HL.HD.TL.HD::PNM->PNM;
        If lengt(HL)=1 then HL.HD.HD::PNM->PNM;close;

end;

FUNCTION ORPM XL;
COMMENT given that some pieces on board are original;

vars    WL BL WHL BHL;
        [WQ WB WN WR]->WL;[BQ BB BN BR]->BL;
loop:   If WL.null then return;
        else dest(WL)->WL->WHL;dest(BL)->BL->BHL;
                If memb("BOTH",XL) and memb(WHL,XL) then WHL::ORP->ORP;
                        WHL::ORP->ORP;
                elseif memb(WHL,XL) then WHL::ORP->ORP;
                close;

                If memb("BOTH",XL) and memb(BHL,XL) then BHL::ORP->ORP;
                        BHL::ORP->ORP;
                elseif memb(BHL,XL) then BHL::ORP->ORP;
                close;
        goto loop;
        close;
end;

FUNCTION ODDM XL;COMMENT        odds were given;

        vars    BL WL BLH WLH ;
        [BQ BN BB BR]->BL;[WQ WN WB WR]->WL;

loop:   If BL.null then return;
        else dest(BL)->BL->BLH;dest(WL)->WL->WLH;
                If memb(BLH,XL) then BLH::ODG->ODG;
                elseif memb(WLH,XL) then WLH::ODG->ODG;
                close;
        goto loop;
        close;
end;



FUNCTION COLM XL;
COMMENT  a piece(s0 have not moved off own colour;

        If memb("QUEEN",XL) and memb("NEITHER",XL) then [WQ BQ]->COLLST;close;
        If memb("WQ",XL) then "WQ"::COLLST->COLLST;close;
        If memb("BQ",XL) then "BQ"::COLLST->COLLST;close;
end;

FUNCTION LMV XL;
COMMENT who moved last;

        If memb("B",XL) or memb("BLACK",XL) then "B"->LM;
        elseif memb("W",XL) or memb("WHITE",XL) then "W"->LM;
        else "U"->LM;
        close;
end;
FUNCTION CAPM XL;
COMMENT captures permitted;
        "N"->CAP;FNINT(XL)->N;
        If N.ISINTEGER then N->CP;
```

```
            else 0->CP;
            close;
    end;
```

```
MMM          MMM    IIIIIIIII    NNN          NNN    PPPPPPPPPPPP
MMM          MMM    IIIIIIIII    NNN          NNN    PPPPPPPPPPPPP
MMM          MMM    IIIIIIIII    NNN          NNN    PPPPPPPPPPPPP
MMMMMMM    MMMMMMM      III      NNN          NNN    PPP         PPP
MMMMMMM    MMMMMMM      III      NNN          NNN    PPP         PPP
MMMMMMM    MMMMMMM      III      NNN          NNN    PPP         PPP
MMM  MMM  MMM           III      NNNNNN       NNN    PPP         PPP
MMM  MMM  MMM           III      NNNNNN       NNN    PPP         PPP
MMM  MMM  MMM           III      NNNNNN       NNN    PPP         PPP
MMM          MMM        III      NNN  NNN     NNN    PPPPPPPPPPPP
MMM          MMM        III      NNN  NNN     NNN    PPPPPPPPPPPPP
MMM          MMM        III      NNN  NNN     NNN    PPPPPPPPPPPPP
MMM          MMM        III      NNN    NNNNNN       PPP
MMM          MMM        III      NNN    NNNNNN       PPP
MMM          MMM        III      NNN       NNN       PPP
MMM          MMM        III      NNN       NNN       PPP
MMM          MMM        III      NNN       NNN       PPP
MMM          MMM    IIIIIIIII    NNN       NNN       PPP
MMM          MMM    IIIIIIIII    NNN       NNN       PPP
MMM          MMM    IIIIIIIII    NNN       NNN       PPP

PPPPPPPPPPPP        OOOOOOOOO      PPPPPPPPPPPP
PPPPPPPPPPPPP       OOOOOOOOO      PPPPPPPPPPPPP
PPPPPPPPPPPPP       OOOOOOOOO      PPPPPPPPPPPPP
PPP         PPP   OOO       OOO    PPP         PPP
PPP         PPP   OOO       OOO    PPP         PPP
PPP         PPP   OOO       OOO    PPP         PPP
PPP         PPP   OOO       OOO    PPP         PPP
PPPPPPPPPPPPP     OOO       OOO    PPPPPPPPPPPPP
PPPPPPPPPPPPP     OOO       OOO    PPPPPPPPPPPPP
PPPPPPPPPPPPP     OOO       OOO    PPPPPPPPPPPPP
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......
PPP               OOO       OOO    PPP                    ......

*START* Job LISTS1 Req #752 for O.ENABLEDOP    Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>MINP.POP.1, created: 28-Mar-84 12:43:48, printed: 11-Jun-86 11:
Job parameters: Request created:11-Jun-86 10:53:57   Page limit:111   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION MINPWN=>BNUM WNUM BHTAB WHTAB;
VARS WHT BHT WN BN;
[%0,0,0%]->BN; [%0,0,0%]->WN;
[%1,2,3,4,5,6,7,8%]->BHT;COPY(BHT)->WHT;
ATHOME(BHT,WHT)->BHT->WHT;
DOCORN("BP",7,6,BN,BHT,-1)->BHT->BN;
DOCORN("WP",2,3,WN,WHT,1)->WHT->WN;
GHOME("BP",7,6,BN,BHT,-1)->BHT->BN;COPYALL(BHT)->BHTAB1;
GHOME("WP",2,3,WN,WHT,1)->WHT->WN;COPYALL(WHT)->WHTAB1;
FFILE("BP",7,BN,BHT,-1)->BHTAB->BNUM;
FFILE("WP",2,WN,WHT,1)->WHTAB->WNUM;
.RSTB;
END;
FUNCTION GHOME PWN RNK NUM NLIST HTAB ADD=>NLIST HTAB;
FORALL I 2 1 7 IF BOARD(I,RNK)=PWN AND BOARD(I+1,RNK)=PWN AND BOARD(I,NUM)=PWN
THEN DELHOM(I-1,RNK,HTAB,NLIST,1)->HTAB->NLIST;PLPH(I-1,NUM,I,PWN,RNK);
CLOSE;CLOSE;
FORALL I 1 1 6 IF BOARD(I,RNK)=PWN AND BOARD(I+2,RNK)=PWN AND BOARD(I+1,NUM)=PWN
FORALL I 1 1 4 IF BOARD(I,RNK)=PWN AND BOARD(I+1,RNK)=PWN AND BOARD(I+3,RNK)=PWN
PLPH(I+2,NUM,I+2,PWN,RNK);CLOSE;CLOSE;
FORALL I 1 1 2 IF BOARD(I,RNK)=PWN AND BOARD(I+1,RNK)=PWN AND BOARD(I+2,RNK)=PWN
BOARD(I+3,NUM)=PWN THEN DELT(I+3,HTAB)->HTAB;PLPH(I+3,NUM,I+3,PWN,RNK);
CLOSE;CLOSE;
END;

FUNCTION RESTB XL YL;
COMMENT to restore board according to XL,YL;

        RESET;copyall(XL)->WLIST;copyall(YL)->BLIST;.PLACE2;

end;
FUNCTION RSTB;
RESET;PLACE(BLIST);PLACE(WLIST);
END;

FUNCTION RSTBO;
COMMENT          restore original board;

        RESET;copyall(NWLIST)->WLIST;copyall(NBLIST)->BLIST;.PLACE2;

end;
FUNCTION PLPH X Y X1 PWN RNK;
"BLANK"->BOARD(X1,Y);PWN->BOARD(X,RNK);
END;
FUNCTION CNTBW I LC NUM CN=>NUM;
VARS H;
IF ERASE((I+LC)//2)=0 THEN NUM.TL.TL.HD->H;H+CN->NUM.TL.TL.HD;
ELSE NUM.TL.HD->H;H+CN->NUM.TL.HD;CLOSE;
NUM.HD->H;H+CN->NUM.HD;
END;
FUNCTION ATHOME BH WH=>WH BH;
VARS I;
FORALL I 1 1 8 IF BOARD(I,2)="WP" THEN DELT(I,WH)->WH;CLOSE;CLOSE;
FORALL I 1 1 8 IF BOARD(I,7)="BP" THEN DELT(I,BH)->BH;CLOSE;CLOSE;
END;
FUNCTION DELT X XL;
IF XL.NULL THEN NIL;
ELSEIF XL.HD=X THEN DELT(X,TL(XL));
ELSE XL.HD::DELT(X,TL(XL));CLOSE;
END;
FUNCTION DELHOM X N HTAB NLIST CP=>NLIST HTAB;
```

```
VARS XL;
DELT(X,HTAB)->HTAB;CNTBW(X,N,NLIST,CP)->NLIST;
END;
FUNCTION DOCORN PWN RNK NUM NNL HTAB ADD=>NNL HTAB;
VARS NNUM XX;NUM->NNUM;
FORALL XX 1 1 6 LPAWN1(XX,RNK,NNL,HTAB,PWN,NUM)->HTAB->NNL;NUM+ADD->NUM;CLOSE;
NNUM->NUM;
FORALL XX 3 1 8 LPAWN2(XX,RNK,NNL,HTAB,PWN,NUM)->HTAB->NNL;NUM+ADD->NUM;CLOSE;
NNUM->NUM;
FORALL XX 2 1 6 LPAWN(XX,RNK,HTAB,PWN,NUM)->HTAB;NUM+ADD->NUM;CLOSE;
NNUM->NUM;
FORALL XX 3 1 7 LPAWNO(XX,RNK,HTAB,PWN,NNUM)->HTAB;NNUM+ADD->NNUM;CLOSE;
END;
FUNCTION LPAWN1 XX RNK NNL HTAB PWN NUM=>NNL HTAB;
VARS FLAG I;0->FLAG;
FORALL I 1 1 XX IF NOT(BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(1,NUM)=PWN THEN DELHOM(XX+1,RNK,HTAB,NNL,XX)->HTAB->NNL;
PLPH(XX+1,NUM,1,PWN,RNK);CLOSE;
END;
FUNCTION LPAWN2 XX RNK NNL HTAB PWN NUM=>NNL HTAB;
VARS FLAG SV I;0->FLAG;11-XX->SV;
FORALL I SV 1 8 IF NOT(BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(8,NUM)=PWN THEN DELHOM(SV-1,RNK,HTAB,NNL,XX-2)->HTAB->NNL;
PLPH(SV-1,NUM,8,PWN,RNK);CLOSE;
END;
FUNCTION LPAWN XX RNK HTAB PWN NUM=>HTAB;
VARS FLAG I;0->FLAG;
FORALL I 2 1 XX IF NOT (BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(1,NUM)=PWN THEN DELT(1,HTAB)->HTAB;PLPH(1,NUM,1,PWN,RNK);CLOSE;
END;
FUNCTION LPAWNO XX RNK HTAB PWN NUM=>HTAB;
VARS FLAG SV I;0->FLAG;10-XX->SV;
FORALL I SV 1 7 IF NOT (BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(8,NUM)=PWN THEN DELT(8,HTAB)->HTAB;PLPH(8,NUM,8,PWN,RNK);CLOSE;
END;
FUNCTION FFILE PWN RNK NUM HTAB AD=>NUM HTAB;
VARS RK I J K;
IF HTAB.NULL THEN EXIT;
IF PWN="BP" THEN 6->RK;ELSE 3->RK;CLOSE;
FORALL K 1 1 8 LUCOL(K,HTAB,PWN,AD,RK,RNK,NUM)->NUM->HTAB;CLOSE;
PLH(HTAB,PWN,RNK,AD)->HTAB;
IF PWN="BP" THEN FORALL I 1 1 8 FORALL J 2 1 6 IF BOARD(I,8-J)="BP"
THEN FNDCAP(PWN,I,8-J,NUM,HTAB,RNK)->NUM->HTAB;CLOSE;CLOSE;CLOSE;
ELSE FORALL I 1 1 8 FORALL J 3 1 7 IF BOARD(I,J)="WP"
THEN FNDCAP(PWN,I,J,NUM,HTAB,RNK)->NUM->HTAB;CLOSE;CLOSE;CLOSE;CLOSE;
END;
FUNCTION LUCOL I HT PN AD RK RNK NUM=>HT NUM;
VARS LC X H CN;RK->LC;
LOOP:LC+AD->LC;
IF LC>7 OR LC<1 THEN RETURN;
ELSEIF NOT(BOARD(I,LC)=PN) THEN GOTO LOOP;
ELSE GABS(I,HT)->X;CLOSE;
IF NOT(ABBS(X-I)=ABBS(LC-RNK)) THEN EXIT;
DELT(X,HT)->HT;PLPH(X,LC,I,PN,RNK);ABBS(X-I)->CN;
CNTBW(I,LC,NUM,CN)->NUM;
END;
FUNCTION PLH HT PN RK AD=>HT;
VARS HL LC XL;COPY(HT)->XL;
LOOP:IF XL.NULL THEN EXIT;
RK->LC;DEST(XL)->XL->HL;
LOOP1:LC+AD->LC;
IF LC>7 OR LC<2 THEN GOTO LOOP;
ELSEIF BOARD(HL,LC)=PN THEN DELT(HL,HT)->HT;PLPH(HL,LC,HL,PN,RK);GOTO LOOP;
ELSE GOTO LOOP1;CLOSE;
```

```
END;
FUNCTION FNDCAP PWN I J NUM HTAB RNK=>HTAB NUM;
VARS CNT X Y XX AD AD2;0->CNT;
GABS(I,HTAB)->X;DELT(X,HTAB)->HTAB;X->XX;
IF X=I THEN RETURN;
ELSEIF PWN="WP" AND X>I THEN -1->AD;1->AD2;2->Y;
ELSEIF PWN="WP" AND X<I THEN 1->AD;1->AD2;2->Y;
ELSEIF PWN="BP" AND X>I THEN -1->AD;-1->AD2;7->Y;
ELSE 1->AD;-1->AD2;7->Y;CLOSE;
LOOP:X+AD->X;Y+AD2->Y;CNT+1->CNT;
IF X=I AND Y=J AND ABBS(X-XX)=ABBS(Y-RNK) THEN CNTBW(I,J,NUM,CNT)->NUM;
ELSEIF X=I OR X<1 OR X>8 OR Y<1 OR Y>8 THEN NUM.HD->H;CNT+H->NUM.HD;
ELSE GOTO LOOP;CLOSE;
END;
FUNCTION GABS I XXL=>X;
        If XXL.null then I->X;exit;
VARS XL;COPY(XXL)->XL;XL.HD->X;
LOOP:IF XL.TL.NULL THEN RETURN;
ELSEIF ABBS(I-XL.TL.HD)<ABBS(I-X)THEN XL.TL.HD->X;CLOSE;
TL(XL)->XL;GOTO LOOP;
END;
```

```
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRRRRRRRRRRR
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRRRRRRRRRRR
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRRRRRRRRRRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR         RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRRRRRRRRRRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRRRRRRRRRRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRRRRRRRRRRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR    RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR    RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR       RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR       RRR
  OOO         OOO       DDD         DDD     DDD         DDD     RRR       RRR
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRR          RRR
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRR          RRR
     OOOOOOOOO          DDDDDDDDDDDD        DDDDDDDDDDDD        RRR          RRR

PPPPPPPPPPPP          OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP          OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP          OOOOOOOOO          PPPPPPPPPPPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPP         PPP     OOO         OOO     PPP         PPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPPPPPPPPPPP        OOO         OOO     PPPPPPPPPPPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP
PPP                 OOO         OOO     PPP                 ......
PPP                 OOO         OOO     PPP                 ......
PPP                 OOO         OOO     PPP                 ......
PPP                   OOOOOOOOO         PPP                 ......
PPP                   OOOOOOOOO         PPP                 ......
PPP                   OOOOOOOOO         PPP                 ......
```

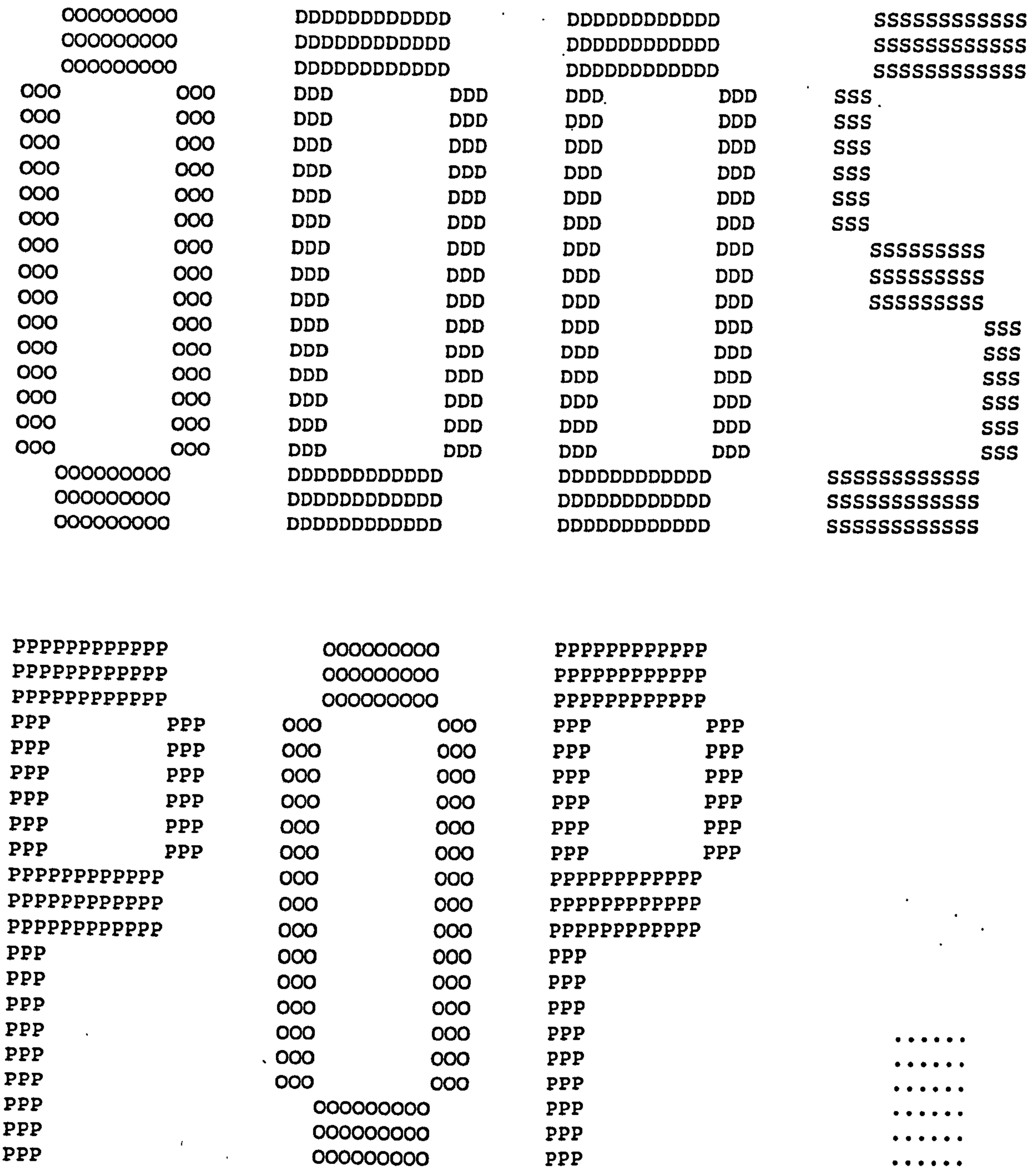
```
*START* Job LISTS1 Req #752 for O.ENABLEDOP    Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>ODDRUL.POP.1, created:  7-Mar-84 12:29:20, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:57   Page limit:111   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION ISPPB;
COMMENT is the promoted piece on board,if yes set global ISPFLG;

        vars ANS;0->ISPFLG;
        .RULE60->ANS;.RULE61->ANS;

end;

FUNCTION ISPBX;
        vars HL;
        If PPAWN="WP" then FCSQ(BDC)->HL;else FCSQ(WDC)->HL;close;
        If lengt(HL)>1 then false;return;
        else [%[IPBRUL],HL,[R64EX]%]::DDB->DDB;true;
        close;
end;

FUNCTION FCSQ XL;
        If XL.null then nil;
        else XL.HD.TL<>FCSQ(TL(XL));
        close;
end;

FUNCTION RULE60;
COMMENT IBPRUL.assumes e.g.[wp 1 2] in prlst.hd.
                if 1 bp captured on its first move and could only have
                captured 1 wp and the wp could not make any captures
                and the home square of the bp is on the wp promotion
                file then the bp captured before the wp promoted;

        vars    PWN XHP BX XHC TP;

        If PPAWN=0 then false;exit;
        PRLST.HD.HD->PWN;PRLST.HD.TL.HD->XHP;
        If PPAWN="WP" then BDC->BX;WLEFTP->TP;WPTT-BTOT->NCAP;
        else WDC->BX;BLEFTP->TP;BPTT-WTOT->NCAP;
        close;

        If BX.null then false;exit;

        BX.HD.HD.TL.HD->XHC;
        If lengt(BX)=1 and lengt(TP)=1 and TP.HD=PWN and NCAP=0 and XHP=XHC
        then true;0->PPONB;1->ISPFLG;[%[ISPRUL],nil,[R60EX]%]::DDB->DDB;
        else false;
        close;
end;

FUNCTION RULE61;
COMMENT IBPRUL.
                If 1 wp captured on its first move and could only have
                captured 1 bp and the bp made n captures in promoting,where
                n is the number of pieces constrained by the wp on its home
                square
                then the wp captured before the bp promoted;

vars    TM PWN XHP YHP TP PN XHC YHC PL NNL LE CT BL WL;
        If PPAWN=0 then false;return;
        elseif PPAWN="WP" then BDC->BX;WLEFTP->TP;
                "BP"->PN;BLIST->PL;
        else WDC->BX;BLEFTP->TP;"WP"->PN;WLIST->PL;
        close;

        If  lengt(BX)/=1 or lengt(TP)/=1 or TP.HD/=PPAWN then false;exit;
        BX.HD->CT;CT.HD.TL.HD->XHP;CT.HD.TL.TL.HD->YHP;
```

```
        CT.TL.HD.HD->XHC;CT.TL.HD.TL.HD->YHC;
        BOARD(XHP,YHP)->TM;UPDATE([%PN,XHC,YHC,XHP,YHP%],PL);
        .CPGTOB->WL->BL;
        If PPAWN="WP" then BL->NNL;else WL->NNL;
        close;

COMMENT how many possible captures by promoting pawn;
        If lengt(XYLIFS)=1 then ABBS(XYLIFS.HD.HD-PRLST.HD.TL.HD)->NCAP;
        else false;return;
        close;

        lengt(NNL)->LE;
        If TM/="BLANK" and  not(memlst(TM,NNL)) then LE-1->LE;close;

        If NCAP=lengt(TP)-LE then true;
        [%[ISPRUL],nil,[R61EX]%]::DDB->DDB;0->PPONB;1->ISPFLG;
        else false;
        close;

COMMENT restore board;
        COPYALL(NWLIST)->WLIST;COPYALL(NBLIST)->BLIST;.RSTB;

end;



FUNCTION WISPP;
COMMENT what is the promoted piece.returns list of possibles
                in PPIECE;

        vars    AH ALR ANS YL;
        copyall(ALLRULES)->ALR;nil->YL;

loop:   If not(ALR.null) then dest(ALR)->ALR->AH;
        If category(valof(AH))="WPPRUL" and eval(premise(valof(AH)))
                then [%[WPPRUL],PPIECE,explain(valof(AH))%]::YL->YL;
                DBL(PPIECE,ORP)->PPIECE;
                close;
                goto loop;
        close;
        If PPIECE.null then return;
        elseif .PPXL then YL<>DDB->DDB;
        [%[WPPRUL],PPIECE,[R74EX]%]::DDB->DDB;
        close;

end;
FUNCTION PPXL;
        vars XL;PPIECE->XL;
loop:   If lengt(XL)=1 then true;
        elseif XL.HD/=XL.TL.HD then false;
        else TL(XL)->XL;goto loop;
        close;
end;



FUNCTION RULE70=>ANS;
COMMENT WIPRUL. what is the promoted piece.
                if ther 3r or 3n or 3b or 2q on board then one of thes
                must be promoted.
        PPIECE is list of possible promoted pieces;

        vars    RK KN QU BI NP;
```

```
        false->ANS;
        If PPAWN=0 then return;
        elseif PPAWN="WP" then "WR"->RK;"WN"->KN;"WQ"->QU;"WB"->BI;
        else "BR"->RK;"BN"->KN;"BQ"->QU;"BB"->BI;
        close;

CONTPC(BI)->NP;If NP>2 then true->ANS;BI::PPIECE->PPIECE;close;
CONTPC(QU)->NP;If NP>1 then true->ANS;QU::PPIECE->PPIECE;close;
CONTPC(KN)->NP;If NP>2 then true->ANS;KN::PPIECE->PPIECE;close;
CONTPC(RK)->NP;If NP>2 then true->ANS;RK::PPIECE->PPIECE;close;
end;

FUNCTION RULE71=>ANS;
COMMENT WPPRULE.given pawn promotion.
                If there >1 bishop on the same colour then one must be
                promoted;

        vars    XL HL LXL BI;
        false->ANS;

        If PPAWN=0 then return;
        elseif PPAWN="WP" then RELCB(WLIST)->XL;"WB"->BI;
        else RELCB(BLIST)->XL;"BB"->BI;
        close;

loop:   If XL.null or lengt(XL)<2 then return;
        else dest(XL)->XL->HL;lengt(XL)->LXL;
        FORALL I 1 1 LXL If CLSQT(HL.HD,HL.TL.HD,XL.HD.HD,XL.HD.TL.HD)
                then true->ANS;BI::PPIECE->PPIECE;close;close;
        goto loop;
        close;
end;

FUNCTION RULE72=>ANS;
COMMENT WPPRUL.
                If there is a bishop on board with its home square constrained
                by pawns then it must be a promoted bishop;

        vars    XL   BL BLX HL BLH X Y BI XLX;
        false->ANS;

        If PPAWN=0 then return;
        elseif PPAWN="WP" then RELCB(WLIST)->XL;WBHSQ->BL;"WB"->BI;
        else RELCB(BLIST)->XL;BBHSQ->BL;"BB"->BI;
        close;

        If  BL.null or XL.null then exit;

        copyall(XL)->XLX;
loop:   If XLX.null then return;
        else dest(XLX)->XLX->HL;HL.HD->X;HL.TL.HD->Y;COPYALL(BL)->BLX;
        close;

loop1:  If BLX.null then goto loop;
        else dest(BLX)->BLX->BLH;
                If CLSQT(BLH.HD,BLH.TL.HD,X,Y) then true->ANS;
                BI::PPIECE->PPIECE;
                close;
        goto loop1;
        close;

end;
```

```
FUNCTION RULE73=>ANS;
COMMENT WPPRUL.
                place all possible pieces on promotion square(s) in PSQS
                and see if can move,reject constrained pieces;

        vars QC BC RU RC XL RK QU KN BI RN QN NN BN PQ PP PPX HL X Y PH
                ML P;
        false->ANS;nil->PP;
        If PPAWN=0 then return;
        elseif PPAWN="WP" then "WR"->RK;"WQ"->QU;"WN"->KN;"WB"->BI;WLIST->XL;
                "W"->P;
        else "BR"->RK;"BQ"->QU;"BN"->KN;"BB"->BI;BLIST->XL;"B"->P;
        close;
        GENCON(P)->QC->BC->RU->RC;

COMMENT collect possible promotion pieces;
        CONTPC(RK)->RN;CONTPC(QU)->QN;CONTPC(KN)->NN;CONTPC(BI)->BN;
        If RN>lengt(RC) then RK::PP->PP;close;
        If QN>lengt(QC) then QU::PP->PP;close;
        If BN>lengt(BC) then BI::PP->PP;close;
        If NN>0 then KN::PP->PP;close;

        copyall(PSQS)->PQ;copyall(PP)->PPX;
loop:   If PQ.null then DELDB(PPIECE)->PPIECE;return;
        else dest(PQ)->PQ->HL;HL.HD->X;HL.TL.HD->Y;
        close;

loop1:  If PPX.null then copyall(PP)->PPX;goto loop;
        else dest(PPX)->PPX->PH;PCMOV([%[%PH,X,Y%]%])->ML;
                If not(ML.null) and CPC(PH) then PH::PPIECE->PPIECE;
                        true->ANS;
                close;
        goto loop1;
        close;


end;

FUNCTION DELDB XL=>CL;
COMMENT delete all doubles from xl ([wn wr wn...]);
        nil->CL;

loop:   If XL.null then return;
        else XL.HD::CL->CL;DELT(XL.HD,XL)->XL;
        goto loop;
        close;

end;


FUNCTION CPC PH;
COMMENT need check here for bishop only
                see if bishop on board  is on colour square as promotion
                square;

        If PH/=BI or BC.null then true;
        elseif lengt(BC)=2 then false;
        else CLSQT(X,Y,BC.HD.TL.HD.HD,BC.HD.TL.HD.TL.HD);
        close;
end;
```



START Job LISTS1 Req #752 for O.ENABLEDOP Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>ODDS.POP.1, created: 4-Apr-84 13:37:37, printed: 11-Jun-86 11:
Job parameters: Request created:11-Jun-86 10:53:57 Page limit:111 Forms:XERO
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:

```
FUNCTION DBL  XXL XL=>YL;
COMMENT deletes all members  of list xxl from list xl,where
                xxl ,xl are of form [a b c ...];
        XXL->YL;
        loop:   If XL.null then return;
                else DELT(XL.HD,YL)->YL;TL(XL)->XL;
                goto loop;
                close;
end;

FUNCTION PRLST XL;
        vars HL;REV(XL)->XL;
        loop:If XL.null then return;
        else XL.HD->HL;nl(1);pr(HL.HD);sp(1);pr("from");sp(1);
        pr("(");pr(HL.TL.TL.TL.HD);pr(",");pr(HL.TL.TL.TL.TL.HD);pr(")");
        sp(1);pr("to");sp(1);pr("(");pr(HL.TL.HD);pr(",");pr(HL.TL.TL.HD);
        pr(")");nl(1);
        close;
        TL(XL)->XL;goto loop;
end;

FUNCTION BCHSQ=>BL WL;
COMMENT returns lists of bishops captured on their home squares;
        nil->BL;nil->WL;
        If BOARD(2,7)="BP" and BOARD(4,7)="BP" and BOARD(3,8)/="BB"
        then [3 8]::BL->BL;close;
        If BOARD(5,7)="BP" and BOARD(7,7)="BP" and BOARD(6,8)/="BB"
        then [6 8]::BL->BL;close;
        If BOARD(5,2)="WP" and BOARD(7,2)="WP" and BOARD(6,1)/="WB"
        then [6 1]::WL->WL;close;
        If BOARD(2,2)="WP" and BOARD(4,2)="WP" and BOARD(3,1)/="WB"
        then [3 1]::WL->WL;close;
end;

FUNCTION COUNTX PI XL;
COMMENT to count the number of pi in xl;
        vars    CB;0->CB;
        loop:If XL.null then CB;return;
                elseif XL.HD=pi THEN 1+CB->CB;
                close;
        TL(XL)->XL;goto loop;
        end;

FUNCTION CONTPC PC=>CB;
COMMENT to count the number of PC in XL - where PC is "wb" etc
                and XL is wlist or blist (pieces on board + locations)
vars XL P;
0->CB;BORW(PC)->P;If P="W" then WLIST->XL;else BLIST->XL;close;
loop: If XL.null then return;
elseif XL.HD.HD=PC then 1+CB->CB;close;
        tl(XL)->XL;goto loop;
end;
FUNCTION PCAPRM PWN LLIFS PSQQ PL LLIFC=>YL;
VARS XS YS XH PN YH NX AD HL ;
PL.HD.TL.HD->XH;PL.HD.HD->PN;PL.HD.TL.TL.HD->YH;NIL->YL;
LLIFS.HD.HD->XS;LLIFS.HD.TL.HD->YS;[%XS,YS%]->XYLIFS;
MEMLST(PSQQ,LLIFC)->HL;IF HL=TRUE THEN EXIT;
IF XH>XS THEN 1->NX;ELSEIF XH<XS THEN -1->NX;ELSE 0->NX;CLOSE;
IF PWN="WP" THEN -1->AD;"BP"->PN;ELSE 1->AD;"WP"->PN;CLOSE;
IF BOARD(XS,YS+AD)=PN AND BOARD(XS+NX,YS+AD)=PN THEN  [%PN,XS+NX,YS+AD,XS+NX,YS%]->HL;
IF PN="BP" THEN UPDATE(HL,BLIST);ELSE UPDATE(HL,WLIST);CLOSE;[%%PWN,XS,YS%]::PNM->PNM;
.SETP;CLOSE;WCAPPN(PSQQ,PWN)->YL;
END;
FUNCTION LROK RO DXL;
```

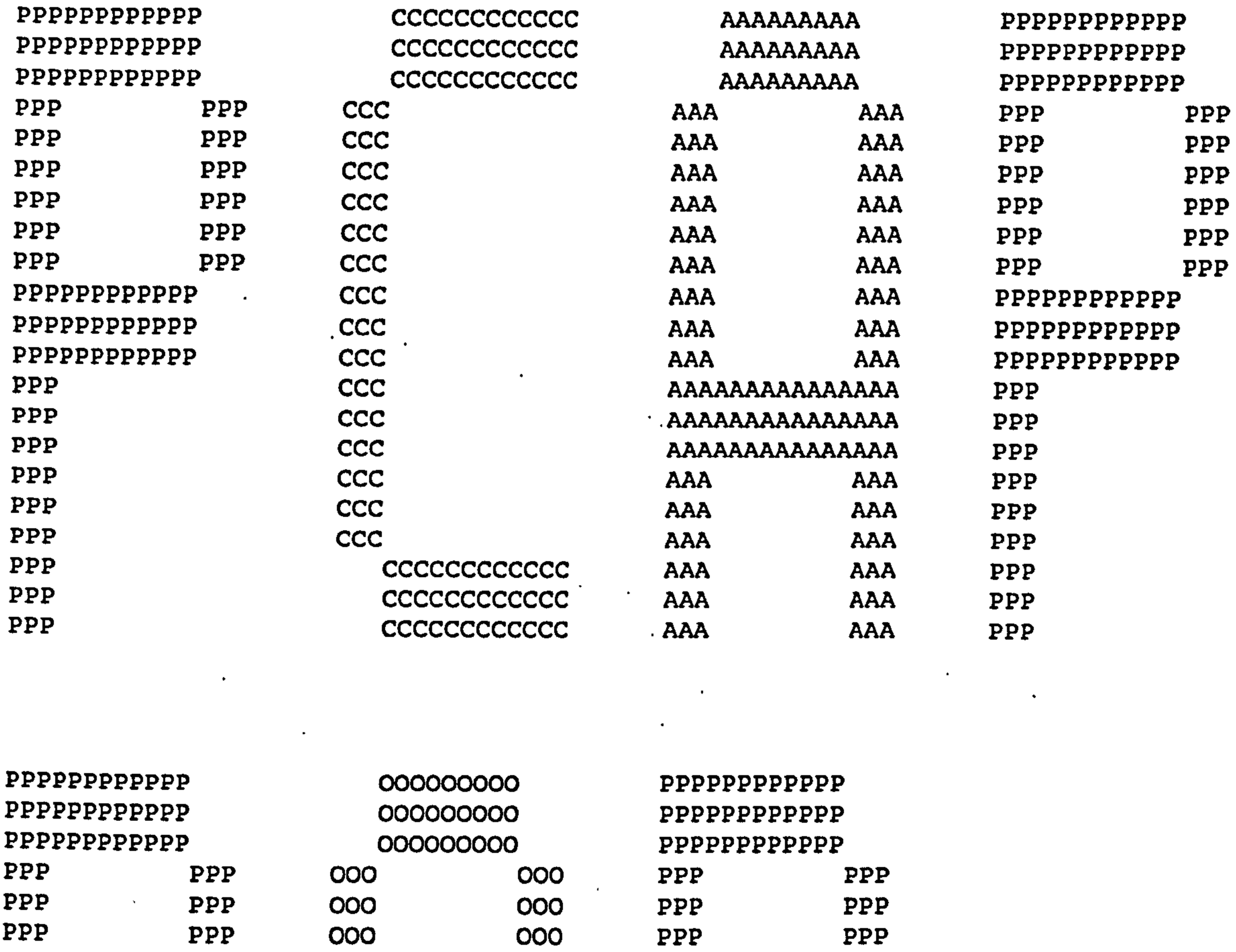
```
IF DXL.NULL THEN TRUE;
ELSEIF BOARD(DXL.HD.HD,DXL.HD.TL.HD)=RO THEN FALSE;
ELSE LROK(RO,TL(DXL));CLOSE;
END;

FUNCTION ASKPP PN;
COMMENT used in wcappn.a b/w pawn has promoted,could it have
                captured a w/b pawn;

        vars    XL NCP YL;

        If PN="WP" then WHTAB->XL;BTOT-WPTT->NCP;[BK 5 8]->YL;
        else BHTAB->XL;WTOT-BPTT->NCP;[WK 1 8]->YL;
        close;

        If lengt(XL)=1 and XL.HD=5 and NCP=0 and memb(YL,PNM) then true;
        else false;
        close;
end;
FUNCTION WCAPPN HL PWN=>YL;
VARS XP YP PN SQ BI ANS CCL P HHL OD RK XS YS  NP;ODG->OD;NIL->YL;
VARS RLC RK TOTN DXL RU RC QC BC YYL LYL YH R1 R2 KL BXY YXP YZ BLISTW WLISTW;
HL.HD->XP;HL->HSQ;HL.TL.HD->YP;BORW(PWN)->P;
        copyall(BLIST)->BLISTW;copyall(WLIST)->WLISTW;
IF PWN="WP" THEN BLEFT->YL;NPB->NP;BTOT-WNUM.HD->NCAP;BTOT->TOTN;"BR"->RK;"BP"->PN;"BB"->BI;
ELSE WLEFT->YL;NPW->NP;WTOT-BNUM.HD->NCAP;WTOT->TOTN;"WR"->RK;"WP"->PN;"WB"->BI;

        If PWN="WP" and YP=8 then DELT("BP",YL)->YL;
                If BPPC or NPB<7 then [BB BN BQ BR]->YL;goto CL;close;
        elseif YP=1 then DELT("WP",YL)->YL;
                If WPPC or NPW<7 then [WB WN WQ WR]->YL;goto CL;close;
        close;

COMMENT check bishops;

        BICOL(YL)->SQ;
        If not(SQ.Null) and lengt(SQ)=1 then D1(BI,YL)->YL;
        elseif not(SQ.null) and CLSQT(XP,YP,SQ.HD,SQ.TL.HD)
                then DELT(BI,YL)->YL;
        close;


        If YL.null then exit;
CL:     COLLST->CCL;
LOOP:IF CCL.NULL THEN GOTO L1;
ELSE BORW(CCL.HD)->P; GSQ(CCL.HD)->HHL;
IF NOT(HHL.NULL) THEN CLSQT(XP,YP,HHL.HD,HHL.TL.HD)->ANS;
IF ANS=FALSE THEN DELT(CCL.HD,YL)->YL;CLOSE;CLOSE;CLOSE;
TL(CCL)->CCL;GOTO LOOP;

COMMENT delete those pieces that place the oppo king in check
                and no way found for this to happen;

L1:     If YP/=8 and YP/=1 then return;
        elseif XYLIFS.null then return;
        else XYLIFS.HD.HD->XS;XYLIFS.HD.TL.HD->YS;
        close;
        YL->YYL;nil->YL;

loop1:If not(YYL.null) then dest(YYL)->YYL->YZ;[%YZ,XP,YP%]->YXP;[%BOARD(XP,YP),XP,YP%]
                If BXY/="BLANK" then BORW(BXY)->P;[%YZ,XP,YP%]->BXY;
                        If P="B" then DELTL(BXY,BLIST)->BLIST;
                        else DELTL(BXY,WLIST)->WLIST;
                        close;
```

```
                close;

                If PWN="WP" then YXP::BLIST->BLIST;
                else YXP::WLIST->WLIST;
                close;

                YZ->BOARD(XP,YP);ISKCHK(BLIST,WLIST)->R1->R2;
                If not(R1=false) and not(R1.null) then KCHECK1(R1,R2)->KL;
                        If not(KL.null) then YZ::YL->YL;close
                elseif not(R1=false) then YZ::YL->YL;
                close;

        NEWARRAY([1 8 1 8],LAMBDA,I,J;"BLANK";END;)->BOARD;
PLACE(BLISTW);PLACE(WLISTW);copyall(BLISTW)->BLIST;copyall(WLISTW)->WLIST;
        goto loop1;
        close;

L2:     D1(PN,YL)->YL;GENCON(P)->QC->BC->RU->RC;
IF RC.NULL THEN DROOK(YL)->YL;EXIT;
COLSQ(RC)->RLC;MEMLST([%XP,YP%],RLC)->ANS;LROK(RK,RLC)->DXL;
IF ANS=TRUE AND XP/=XS AND ABBS(HSQ.HD-XS)=TOTN THEN NIL->YL;
ELSEIF DXL=FALSE OR NCAP/=TOTN OR ANS=TRUE THEN RETURN;ELSE [%RK%]->YL;CLOSE;
END;
FUNCTION DROOK YL=>YL;
VARS ANS1 ANS2 ANS3 XPS;IF XYLIFS.HD/=1 AND XYLIFS.HD/=5 AND XYLIFS.HD/=8 THEN EXIT;
XYLIFS.HD->XPS;
IF PWN="BP" THEN GOTO L1;CLOSE;
MEMLST([BR 1 8],PNM)->ANS1;MEMLST([BK 5 8],PNM)->ANS2;MEMLST([BR 8 8],PNM)->ANS3;
IF XPS=1 OR XPS=5 AND ANS1=FALSE AND ANS2=FALSE AND BOARD(2,7)="BP" AND BOARD(3,7)="BP"
ELSEIF XPS=8 AND ANS2=FALSE AND ANS3=FALSE AND BOARD(6,7)="BP" AND BOARD(7,7)="BP"
L1:MEMLST([WR 1 1],PNM)->ANS1;MEMLST([WK 5 1],PNM)->ANS2;MEMLST([WR 8 1],PNM)->ANS3;
IF XPS=1 OR XPS=5 AND ANS1=FALSE AND ANS2=FALSE AND BOARD(2,2)="WP" AND BOARD(3,2)="WP"
ELSEIF XPS=8 AND ANS2=FALSE AND ANS3=FALSE AND BOARD(5,2)="WP" AND BOARD(6,2)="WP"
END;
FUNCTION COLSQ RC;
IF RC.NULL THEN NIL;
ELSE RC.HD.TL<>COLSQ(TL(RC));CLOSE;
END;
FUNCTION CPBECP HT PWN=>CL;
VARS BD WD XL CQ NC CQQ RNK;NIL->CL;
.CAPSQ->BD->WD;
IF PWN="WP" THEN BD->XL;2->RNK;BTOT-WNUM.HD->NC;
ELSE WD->XL;7->RNK;WTOT-BNUM.HD->NC;CLOSE;
CSQ(XL)->CQ;COPYALL(CQ)->CQQ;
LOOP1:IF HT.NULL THEN EXIT;COPYALL(CQQ)->CQ;
LOOP:IF CQ.NULL THEN TL(HT)->HT;GOTO LOOP1;
ELSEIF ABBS(HT.HD-CQ.HD)>NC THEN [%PWN,HT.HD,RNK%]::CL->CL;RETURN;CLOSE;
CQ.TL->CQ;GOTO LOOP;
END;
FUNCTION LENGT XL=>LE;
IF XL.NULL THEN 0->LE;ELSE LENGTH(XL)->LE;CLOSE;
END;
FUNCTION CASQ PN HHL=>CSQ;
VARS HL XL;NIL->CSQ;
IF PN="WP" THEN BCAS->XL;ELSE WCAS->XL;CLOSE;
LOOP:IF XL.NULL THEN MEMLST(HHL,CSQ)->CSQ;EXIT;
XL.HD.HD->HL;[%HL.TL.HD,HL.TL.TL.HD%]::CSQ->CSQ;
XL.HD.TL.HD->HL;[%HL.TL.HD,HL.TL.TL.HD%]::CSQ->CSQ;
TL(XL)->XL;GOTO LOOP;
END;
FUNCTION POSCAS=>WC BC;
VARS BL;"BLANK"->BL;NIL->WC;NIL->BC;
IF BOARD(5,8)/="BK" THEN GOTO L1;CLOSE;
IF BOARD(1,8)="BR" AND BOARD(2,8)=BL AND BOARD(3,8)=BL AND BOARD(4,8)=BL
```

```
THEN [[BR 1 8][BK 5 8]]::BC->BC;CLOSE;
IF BOARD(6,8)=BL AND BOARD(7,8)=BL AND BOARD(8,8)="BR"
THEN [[BR 8 8][BK 5 8]]::BC->BC;CLOSE;
L1:IF BOARD(5,1)/="WK" THEN EXIT;
IF BOARD(1,1)="WR" AND BOARD(2,1)=BL AND BOARD(3,1)=BL AND BOARD(4,1)=BL
THEN [[WR 1 1][WK 5 1]]::WC->WC;CLOSE;
IF BOARD(6,1)=BL AND BOARD(7,1)=BL AND BOARD(8,1)="WR"
THEN [[WR 8 1][WK 5 1]]::WC->WC;CLOSE;
END;
FUNCTION PLACE2;
PLACE(BLIST);PLACE(WLIST);
END;

FUNCTION ADDNUM P;
        If P="BP" then BPTT+2->BPTT;
        else WPTT+2->WPTT;
        close;
end;

FUNCTION SETP;
COMMENT this function sets up position variables;
VARS PC BL WL;
COMMENT castle tables. e.g.  [[[wr 1 1][wk 5 1]][wr 8 1][wk 5 1]]];
.POSCAS->BCAS->WCAS;

COMMENT whcap blcap are lists of captured pieces.
                wl bl are preliminary lists of pieces captured on board;
.CPLIST->WHCAP->BLCAP;.CPGTOB->WL->BL;

COMMENT whtab/bhtab are lists of available pawn home squars.
                wnum/bnum = [x y z] whrer x=total pawn captures
                        y=captures on white squares,z=captures on black;
.MINPWN->WHTAB->BHTAB->WNUM->BNUM;

COMMENT from pieces captured on board delete those pieces given as odds
        resultant lists in wleft/bleft;
IF NOT(ODG.NULL) THEN BORW(ODG.HD)->PC;
IF PC="W" THEN DBL(WL,ODG)->WLEFT;BL->BLEFT;ELSE DBL(BL,ODG)->BLEFT;WL->WLEFT;CLOSE;
ELSE BL->BLEFT;WL->WLEFT;CLOSE;

COMMENT if a pawn has promoted delete from possible capture list;

        If PPAWN="BP" then D1("BP",BLEFT)->BLEFT;
        elseif PPAWN="WP" then D1("WP",WLEFT)->WLEFT;
        close;

LENGT(WLEFT)->WTOT;LENGT(BLEFT)->BTOT;
COMMENT total no.of pawn captures in wpt/bpt wptt/bptt
        need 2 - truoble with wpt/bpt;
WNUM.HD->WPT;BNUM.HD->BPT;
WNUM.HD->WPTT;BNUM.HD->BPTT;

COMMENT no.of pawns on board in npb/npw;
NPWN("B")->NPB;NPWN("W")->NPW;

COMMENT find pawn capture squares e.eg. [[[wp 8  2][7 3]][[wp 5 2][6 3][7 4][8 5]]]
        place in wdc/bdc;
        .CAPSQ->BDC->WDC;
COMMENT look for cross-captures and add to list;

LFXCAP("BP")->XCAPB;LFXCAP("WP")->XCAPW;
IF NOT(XCAPB.NULL) THEN XCAPB<>BDC->BDC;ADDNUM("BP");CLOSE;
IF NOT(XCAPW.NULL) THEN XCAPW<>WDC->WDC;ADDNUM("WP");CLOSE;
```

```
COMMENT cpbx/cpwx gives x co-ords of capture squares only
                cpb /cpw gives capture squares only;
CSQ(BDC)->CPBX;CSQ(WDC)->CPWX;CSQ1(BDC)->CPB;CSQ1(WDC)->CPW;

COMMENT         in wleftp/bleftp form lists of pieces that could be captured
                by pawns;
IF CPB.NULL THEN WLEFT->WLEFTP;
ELSEIF BNUM.TL.HD=0 OR BNUM.TL.TL.HD=0 THEN WCAPPN(CPB.HD,"BP")->WLEFTP;
ELSE WLEFT->WLEFTP;CLOSE;
IF CPW.NULL THEN BLEFT->BLEFTP;
ELSEIF WNUM.TL.HD=0 OR WNUM.TL.TL.HD=0 THEN WCAPPN(CPW.HD,"WP")->BLEFTP;
ELSE BLEFT->BLEFTP;CLOSE;

COMMENT wbhsq/bbhsq are lists of squares of bishops captured at home;
.BCHSQ->WBHSQ->BBHSQ;
END;
FUNCTION LFXCAP PWN=>XC;
VARS RNK FL;NIL->XC;
IF PWN="BP" THEN 7->RNK;6->FL;WTOT-BPTT->NCAP;
ELSE 2->RNK;3->FL;BTOT-WPTT->NCAP;CLOSE;
IF NCAP>1 THEN FORALL I 1 1 7 IF BOARD(I,FL)=PWN AND BOARD(I+1,FL)=PWN
        then [%[%PWN,I,RNK%],[%I+1,FL%]%]::XC->XC;
             [%[%PWN,I+1,RNK%],[%I,FL%]%]::XC->XC;
        close;close;close;
END;
FUNCTION NPIECES=>BN WN;
CLIST(BLIST)->BN;CLIST(WLIST)->WN;
END;
FUNCTION COPYALL P=>P1;
IF P.ISLIST AND P/=NIL THEN CONS(P.HD.COPYALL,P.TL.COPYALL);ELSE P;
CLOSE->P1;
END;
FUNCTION NPWN BW=>CNT;
VARS XL PN;0->CNT;
IF BW="B" THEN BLIST->XL;"BP"->PN;
ELSE WLIST->XL;"WP"->PN;CLOSE;
LOOP:IF XL.NULL THEN RETURN;
ELSEIF XL.HD.HD=PN THEN CNT+1->CNT;CLOSE;
TL(XL)->XL;GOTO LOOP;
END;

FUNCTION CPLIST=>LB LW;
VARS PL;
[BR BR BB BB BN BN BK BQ BP BP BP BP BP BP BP BP]->PL;
DELCAP(PL,BLIST)->LB;
[WR WR WB WB WN WN WK WQ WP WP WP WP WP WP WP WP]->PL;
DELCAP(PL,WLIST)->LW;
END;
FUNCTION DELCAP XL BL=>XL;
VARS HL CL LI ANS;BL->CL;
LOOP:IF CL.NULL THEN RETURN;
ELSE DEST(CL)->CL->HL;CLOSE;
NIL->LI;MEMLST(HL.HD,XL)->ANS;IF ANS=FALSE THEN CAPDEL(XL,HL,LI)->XL;CLOSE;
GOTO LOOP;
END;
FUNCTION CAPDEL X H LI=>LI;
LOOP:IF H.HD=X.HD THEN LI<>TL(X)->LI;
ELSE X.HD::LI->LI;TL(X)->X;GOTO LOOP;CLOSE;
END;
FUNCTION PLOCB XL=>XL;
VARS PC XP YP HL BW;
XL.HD->HL;HL.HD->PC;HL.TL.HD->XP;HL.TL.TL.HD->YP;
PC->BOARD(XP,YP);TL(XL)->XL;
BORW(PC)->BW;IF BW="W" THEN HL::WLIST->WLIST;ELSE HL::BLIST->BLIST;CLOSE;
```

```
.SETP;
END;
FUNCTION BORW PC;
IF PC="WP" OR PC="WB" OR PC="WN" OR PC="WQ" OR PC="WK" OR PC="WR"
THEN "W";ELSE "B";CLOSE;
END;
FUNCTION CLIST XL;
VARS N;0->N;
L1:IF XL.NULL THEN N;EXIT;
N+1->N;TL(XL)->XL;GOTO L1;
END;
MACRO RESET;
"BLANK"->BLANK;
  MACRESULTS([%
    "NEWARRAY","(","[",1,8,1,8,"]",",",
    "LAMBDA","I","J",";",BLANK,
     ";","END",";",")","->","BOARD",";"
           %])
END;
COMMENT function from legality rules - useful to use by itself;

FUNCTION CHPWN R1;
IF R1.NULL THEN TRUE;
ELSEIF R1.HD.TL.HD="WP" AND R1.HD.TL.TL.HD=2 THEN FALSE;
ELSEIF R1.HD.TL.HD="BP" AND R1.HD.TL.TL.HD=7 THEN FALSE;
ELSE CHPWN(TL(R1));CLOSE;
END;
```



START Job LISTS1 Req #752 for O.ENABLEDOP Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>PCAPSQ.POP.1, created: 16-Sep-83 13:18:15, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:57 Page limit:111 Forms:XERO
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:

```
FUNCTION CAPSQ=>WDC BDC;
VARS WHT BHT WN BN WDC BDC;NIL->WDC;NIL->BDC;
[%0,0,0%]->BN; [%0,0,0%]->WN;
[%1,2,3,4,5,6,7,8%]->BHT;COPY(BHT)->WHT;
ATHOME(BHT,WHT)->BHT->WHT;
DOCRN("BP",7,6,BN,BHT,-1)->BHT->BN;
DOCRN("WP",2,3,WN,WHT,1)->WHT->WN;
GHOME1("BP",7,6,BN,BHT,-1)->BHT->BN;
GHOME1("WP",2,3,WN,WHT,1)->WHT->WN;
FFILE1("BP",7,BN,BHT,-1);
FFILE1("WP",2,WN,WHT,1);
.RSTB;
END;
FUNCTION GHOME1 PWN RNK NUM NLIST HTAB ADD=>NLIST HTAB;
FORALL I 2 1 7 IF BOARD(I,RNK)=PWN AND BOARD(I+1,RNK)=PWN AND BOARD(I,NUM)=PWN
THEN DELHM1(I-1,RNK,HTAB,NLIST,1)->HTAB->NLIST;PLPH(I-1,NUM,I,PWN,RNK);
CLOSE;CLOSE;
FORALL I 1 1 6 IF BOARD(I,RNK)=PWN AND BOARD(I+1,RNK)=PWN AND BOARD(I+1,NUM)=PWN
THEN DELHM1(I+2,RNK,HTAB,NLIST,1)->HTAB->NLIST;PLPH(I+2,NUM,I+1,PWN,RNK);
CLOSE;CLOSE;
END;
FUNCTION CNTBW1 I LC NUM CN PWN=>NUM;
VARS H;
IF ERASE((I+LC)//2)=0 THEN NUM.TL.TL.HD->H;H+CN->NUM.TL.TL.HD;
ELSE NUM.TL.HD->H;H+CN->NUM.TL.HD;CLOSE;
NUM.HD->H;H+CN->NUM.HD;
LPCAPS(I,LC,CN,PWN);
END;
FUNCTION DELT X XL;
IF XL.NULL THEN NIL;
ELSEIF XL.HD=X THEN DELT(X,TL(XL));
ELSE XL.HD::DELT(X,TL(XL));CLOSE;
END;
FUNCTION DELHM1 X N HTAB NLIST CP=>NLIST HTAB;
VARS XL;
DELT(X,HTAB)->HTAB;CNTBW1(X,N,NLIST,CP,PWN)->NLIST;
END;
FUNCTION DOCRN PWN RNK NUM NNL HTAB ADD=>NNL HTAB;
VARS NNUM XX;NUM->NNUM;
FORALL XX 1 1 6 LPWN1(XX,RNK,NNL,HTAB,PWN,NUM)->HTAB->NNL;NUM+ADD->NUM;CLOSE;
NNUM->NUM;
FORALL XX 3 1 8 LPWN2(XX,RNK,NNL,HTAB,PWN,NUM)->HTAB->NNL;NUM+ADD->NUM;CLOSE;
END;
FUNCTION LPWN1 XX RNK NNL HTAB PWN NUM=>NNL HTAB;
VARS FLAG I;0->FLAG;
FORALL I 1 1 XX IF NOT(BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(1,NUM)=PWN THEN DELHM1(XX+1,RNK,HTAB,NNL,XX)->HTAB->NNL;
PLPH(XX+1,NUM,1,PWN,RNK);CLOSE;
END;
FUNCTION LPWN2 XX RNK NNL HTAB PWN NUM=>NNL HTAB;
VARS FLAG SV I;0->FLAG;11-XX->SV;
FORALL I SV 1 8 IF NOT(BOARD(I,RNK)=PWN) THEN 1->FLAG;CLOSE;CLOSE;
IF FLAG=0 AND BOARD(8,NUM)=PWN THEN DELHM1(SV-1,RNK,HTAB,NNL,XX-2)->HTAB->NNL;
PLPH(SV-1,NUM,8,PWN,RNK);CLOSE;
END;
FUNCTION FFILE1 PWN RNK NUM HTAB AD;
VARS RK COPH I J K;
IF HTAB.NULL THEN EXIT;
IF PWN="BP" THEN 6->RK;ELSE 3->RK;CLOSE;
FORALL K 1 1 8 LUCOL1(K,HTAB,PWN,AD,RK,RNK,NUM)->NUM->HTAB;CLOSE;
COPY(HTAB)->COPH;PLH(HTAB,PWN,RNK,AD)->HTAB;
IF PWN="BP" THEN FORALL I 1 1 8 FORALL J 2 1 6 IF BOARD(I,8-J)="BP"
THEN FNDCP1(PWN,I,8-J,NUM,HTAB,RNK)->NUM->HTAB;CLOSE;CLOSE;CLOSE;
ELSE FORALL I 1 1 8 FORALL J 3 1 7 IF BOARD(I,J)="WP"
```

```
THEN FNDCP1(PWN,I,J,NUM,HTAB,RNK)->NUM->HTAB;CLOSE;CLOSE;CLOSE;CLOSE;
COPY(COPH)->HTAB;
END;
FUNCTION LUCOL1 I HT PN AD RK RNK NUM=>HT NUM;
VARS LC X H CN;RK->LC;
LOOP:LC+AD->LC;
IF LC>7 OR LC<1 THEN RETURN;
ELSEIF NOT(BOARD(I,LC)=PN) THEN GOTO LOOP;
ELSE GABS(I,HT)->X;CLOSE;
IF NOT(ABBS(X-I)=ABBS(LC-RNK)) THEN EXIT;
ABBS(X-I)->CN;DELHM1(X,RNK,HT,NUM,CN)->HT->NUM;PLPH(X,LC,I,PN,RNK);
END;
FUNCTION FNDCP1 PWN I J NUM HTAB RNK=>HTAB NUM;
VARS CNT X Y XX AD AD2;0->CNT;
GABS(I,HTAB)->X;DELT(X,HTAB)->HTAB;X->XX;
IF X=I THEN RETURN;
ELSEIF PWN="WP" AND X>I THEN -1->AD;1->AD2;2->Y;
ELSEIF PWN="WP" AND X<I THEN 1->AD;1->AD2;2->Y;
ELSEIF PWN="BP" AND X>I THEN -1->AD;-1->AD2;7->Y;
ELSE 1->AD;-1->AD2;7->Y;CLOSE;
LOOP:X+AD->X;Y+AD2->Y;CNT+1->CNT;
IF X=I AND Y=J AND ABBS(X-XX)=ABBS(Y-RNK) THEN CNTBW1(XX,RNK,NUM,CNT,PWN)->NUM;
ELSEIF X=I OR X<1 OR X>8 OR Y<1 OR Y>8 THEN NUM.HD->H;CNT+H->NUM.HD;
ELSE GOTO LOOP;CLOSE;
END;
FUNCTION LPCAPS X N CP PWN;
VARS XP YP ADD XA PL AD;NIL->PL;
IF PWN="BP" THEN -1->ADD;1->AD;ELSE 1->ADD;-1->AD;CLOSE;
X+CP->XP;N+(CP*ADD)->YP;
IF XP>8 THEN X-CP->XP;
ELSEIF NOT(BOARD(XP,YP)=PWN) THEN X-CP->XP;CLOSE;
IF XP>X THEN -1->XA;ELSE 1->XA;CLOSE;
LOOP:[%XP,YP%]::PL->PL;YP+AD->YP;XP+XA->XP;
CP-1->CP;IF NOT(CP=0) THEN GOTO LOOP;
ELSE [%PWN,X,N%]::PL->PL;
IF PWN="BP" THEN PL::BDC->BDC;ELSE PL::WDC->WDC;CLOSE;CLOSE;
END;
```

```
RRRRRRRRRRR          EEEEEEEEEEEEEEEE  VVV       VVV  MMM          MMM
RRRRRRRRRRRR         EEEEEEEEEEEEEEEE  VVV       VVV  MMM          MMM
RRRRRRRRRRRR         EEEEEEEEEEEEEEEE  VVV       VVV  MMM          MMM
RRR          RRR     EEE               VVV       VVV  MMMMMMM      MMMMMMM
RRR          RRR     EEE               VVV       VVV  MMMMMMM      MMMMMMM
RRR          RRR     EEE               VVV       VVV  MMMMMMM      MMMMMMM
RRR          RRR     EEE               VVV       VVV  MMM  MMM      MMM
RRR          RRR     EEE               VVV       VVV  MMM  MMM      MMM
RRR          RRR     EEE               VVV       VVV  MMM  MMM      MMM
RRRRRRRRRRR          EEEEEEEEEEE       VVV       VVV  MMM          MMM
RRRRRRRRRRRR         EEEEEEEEEEEE       VVV     VVV   MMM          MMM
RRRRRRRRRRRR         EEEEEEEEEEEEE      VVV     VVV   MMM          MMM
RRR     RRR          EEE                 VVV   VVV    MMM          MMM
RRR     RRR          EEE                 VVV   VVV    MMM          MMM
RRR     RRR          EEE                 VVV   VVV    MMM          MMM
RRR     RRR          EEE                 VVV   VVV    MMM          MMM
RRR         RRR      EEEEEEEEEEEEEEEE      VVV        MMM          MMM
RRR         RRR      EEEEEEEEEEEEEEEE      VVV        MMM          MMM
RRR         RRR      EEEEEEEEEEEEEEEE      VVV        MMM          MMM

PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPPP           OOOOOOOOO          PPPPPPPPPPPPP
PPPPPPPPPPPPP           OOOOOOOOO          PPPPPPPPPPPPP
PPP          PPP      OOO         OOO      PPP          PPP
PPP          PPP     .OOO         OOO      PPP          PPP
PPP          PPP      OOO         OOO      PPP          PPP
PPP          PPP      OOO         OOO      PPP          PPP
PPP          PPP      OOO         OOO      PPP          PPP
PPPPPPPPPPPP          OOO         OOO      PPPPPPPPPPPP
PPPPPPPPPPPPP         OOO         OOO      PPPPPPPPPPPPP
PPPPPPPPPPPPP         OOO         OOO      PPPPPPPPPPPPP
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......
PPP                   OOO         OOO      PPP                    ......

*START* Job LISTS1 Req #752 for O.ENABLEDOP    Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>REVMOV.POP.1, created:  5-Aug-83 14:35:51, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:57   Page limit:111   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION PCMOV LIST=>MOVLST;
COMMENT          REVERSE MOVE GENERATOR."LIST" IS THE LIST OF PIECES ON
                 BOARD TOGETHER WITH THEIR LOCATIONS;
VARS XL HL LI HED PC XP YP;NIL->MOVLST;LIST->XL;
LOOP:IF XL.NULL THEN EXIT;
DEST(XL)->XL->HL;
HL.HD->PC;HL.TL.HD->XP;HL.TL.TL.HD->YP;
IF PC="WR" OR PC="BR" THEN RMOVE(PC,XP,YP);
ELSEIF PC="WB" OR PC="BB" THEN BMOVE(PC,XP,YP);
ELSEIF PC="WN" OR PC="BN" THEN NMOVE(PC,XP,YP);
ELSEIF PC="WQ" OR PC="BQ" THEN QMOVE(PC,XP,YP);
ELSEIF PC="WK" OR PC="BK" THEN KMOVE(PC,XP,YP);
ELSE PMOVE(PC,XP,YP);
CLOSE;
->LI;
IF NOT(NULL(LI)) THEN LI<>MOVLST->MOVLST;CLOSE;
GOTO LOOP;
END;

FUNCTION KMOVE PC XP YP=>LI;
COMMENT KING REVERSE MOVES;
VARS XL HE;
NIL->LI;
[%[%XP-1,YP-1%],[%XP-1,YP%],[%XP-1,YP+1%],[%XP,YP-1%],[%XP,YP+1%],[%XP+1,YP-1%]
,[%XP+1,YP%],[%XP+1,YP+1%]%]->XL;
LOOP:IF NULL(XL) THEN EXIT; DEST(XL)->XL->HE;
IF HE.HD>=1 AND HE.HD=<8 AND HE.TL.HD>=1 AND HE.TL.HD=<8 AND BOARD(HE.HD,HE.TL.HD)='BLANK'
GOTO LOOP;                    THEN [%PC,XP,YP,HE.HD,HE.TL.HD%]::LI=>LI;
END;

FUNCTION NMOVE PC XP YP=>LI;
COMMENT          KNIGHT REVERSE MOVES;
VARS XL HL;NIL->LI;
[%[%XP+2,YP+1%],[%XP+2,YP-1%],[%XP+1,YP+2%],[%XP+1,YP-2%],[%XP-1,YP+2%],
[%XP-2,YP+1%],[%XP-1,YP-2%],[%XP-2,YP-1%]%]->XL;
LOOP: IF NULL(XL) THEN EXIT;
DEST(XL)->XL->HL;
IF HL.HD>=1 AND HL.HD=<8 AND HL.TL.HD>=1 AND HL.TL.HD=<8 AND BOARD(HL.HD,HL.TL.HD)='BLANK'
GOTO LOOP;                    THEN [%PC,XP,YP,HL.HD,HL.TL.HD%1::LI=>LI;
END;

FUNCTION RMOVE PC XP YP=>LI;
COMMENT ROOK REVERSE MOVES;
VARS XS YS;
YP->YS;XP->XS;NIL->LI;
LOOP1: YS+1->YS;
IF YS>8 OR NOT(BOARD(XP,YS)="BLANK") THEN YP->YS;GOTO LOOP2;
ELSE [%PC,XP,YP,XP,YS%]::LI->LI;GOTO LOOP1;CLOSE;
LOOP2: YS-1->YS;
IF YS<1 OR NOT(BOARD(XP,YS)="BLANK") THEN YP->YS;GOTO LOOP3;
ELSE [%PC,XP,YP,XP,YS%]::LI->LI;GOTO LOOP2;CLOSE;
LOOP3:XS+1->XS;
IF XS>8 OR NOT(BOARD(XS,YP)="BLANK") THEN XP->XS;GOTO LOOP4;
ELSE [%PC,XP,YP,XS,YP%]::LI->LI;GOTO LOOP3;CLOSE;
LOOP4:XS-1->XS;
IF XS<1 OR NOT(BOARD(XS,YP)="BLANK") THEN RETURN;
ELSE [%PC,XP,YP,XS,YP%]::LI->LI;GOTO LOOP4;CLOSE;
END;

FUNCTION PMOVE PC XP YP=>LI;
COMMENT PAWN REVERSE MOVES;
NIL->LI;
IF PC="WP" AND YP=2 THEN RETURN;
ELSEIF PC="BP" AND YP=7 THEN RETURN;CLOSE;
```

```
IF PC="WP" AND YP=4 AND BOARD(XP,2)="BLANK" AND BOARD(XP,3)="BLANK"
THEN [%PC,XP,YP,XP,2%]::LI->LI;CLOSE;
IF PC="WP" AND BOARD(XP,YP-1)="BLANK" THEN [%PC,XP,YP,XP,YP-1%]::LI->LI;CLOSE;
IF PC="WP" AND (XP-1)>0 AND BOARD(XP-1,YP-1)="BLANK" THEN [%PC,XP,YP,XP-1,YP-1%]::LI->LI;close
IF PC="WP" AND (XP+1)<8 AND BOARD(XP+1,YP-1)="BLANK"THEN [%PC,XP,YP,XP+1,YP-1%]::LI->LI;close
IF PC="BP" AND YP=5 AND BOARD(XP,7)="BLANK"  AND BOARD(XP,6)="BLANK"
THEN [%PC,XP,YP,XP,7%]::LI->LI;CLOSE;
IF PC="BP" AND BOARD(XP,YP+1)="BLANK" THEN [%PC,XP,YP,XP,YP+1%]::LI->LI;CLOSE;
IF PC="BP" AND (XP-1)>0 AND BOARD(XP-1,YP+1)="BLANK" THEN [%PC,XP,YP,XP-1,YP+1%]::LI->LI;
IF PC="BP" AND (XP+1)<9 AND BOARD(XP+1,YP+1)="BLANK"                                    close.
 THEN [%PC,XP,YP,XP+1,YP+1%]::LI->LI;CLOSE;
END;

FUNCTION BMOVE PC XP YP=>LI;
COMMENT BISHOP REVERSE MOVES;
VARS XS YS;NIL->LI;
XP->XS;YP->YS;
LOOP1:XS+1->XS;YS+1->YS;
IF XS=<8 AND YS=<8 AND BOARD(XS,YS)="BLANK"
THEN [%PC,XP,YP,XS,YS%]::LI->LI;GOTO LOOP1;CLOSE;
XP->XS;YP->YS;
LOOP2:XS-1->XS;YS+1->YS;
IF XS>=1 AND YS=<8 AND BOARD(XS,YS)="BLANK"
THEN [%PC,XP,YP,XS,YS%]::LI->LI;GOTO LOOP2;CLOSE;
XP->XS;YP->YS;
LOOP3:XS-1->XS;YS-1->YS;
IF XS>=1 AND YS>=1 AND BOARD(XS,YS)="BLANK"
THEN [%PC,XP,YP,XS,YS%]::LI->LI;GOTO LOOP3;CLOSE;
XP->XS;YP->YS;
LOOP4:XS+1->XS;YS-1->YS;
IF XS=<8 AND YS>=1 AND BOARD(XS,YS)="BLANK"
THEN [%PC,XP,YP,XS,YS%]::LI->LI;GOTO LOOP4;CLOSE;
END;

FUNCTION QMOVE PC XP YP=>LI;
COMMENT QUEEN REVERSE MOVES;
VARS XL;
RMOVE(PC,XP,YP)->XL;XL->LI;
BMOVE(PC,XP,YP)->XL;XL<>LI->LI;
END;
```

```
RRRRRRRRRRR          EEEEEEEEEEEEEEE      XXX          XXX      PPPPPPPPPPP
RRRRRRRRRRRR         EEEEEEEEEEEEEEE      XXX          XXX      PPPPPPPPPPPP
RRRRRRRRRRRR         EEEEEEEEEEEEEEE      XXX          XXX      PPPPPPPPPPPP
RRR          RRR     EEE                  XXX          XXX      PPP          PPP
RRR          RRR     EEE                  XXX          XXX      PPP          PPP
RRR          RRR     EEE                  XXX          XXX      PPP          PPP
RRR          RRR     EEE                     XXX    XXX         PPP          PPP
RRR          RRR     EEE                     XXX    XXX         PPP          PPP
RRR          RRR     EEE                     XXX    XXX         PPP          PPP
RRRRRRRRRRR          EEEEEEEEEEEE               XXX              PPPPPPPPPPP
RRRRRRRRRRRR         EEEEEEEEEEEE               XXX              PPPPPPPPPPPP
RRRRRRRRRRRR         EEEEEEEEEEEE               XXX              PPPPPPPPPPPP
RRR    RRR           EEE                     XXX    XXX         PPP
RRR    RRR           EEE                     XXX    XXX         PPP
RRR    RRR           EEE                        XXX    XXX      PPP
RRR       RRR        EEE                  XXX          XXX      PPP
RRR       RRR        EEE                  XXX          XXX      PPP
RRR       RRR        EEE                  XXX          XXX      PPP
RRR          RRR     EEEEEEEEEEEEEEE      XXX          XXX      PPP
RRR          RRR     EEEEEEEEEEEEEEE      XXX          XXX      PPP
RRR          RRR     EEEEEEEEEEEEEEE      XXX          XXX      PPP
```

```
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO          PPPPPPPPPPPP
PPP          PPP      OOO        OOO       PPP          PPP
PPP          PPP      OOO        OOO       PPP          PPP
PPP          PPP      OOO        OOO       PPP          PPP
PPP          PPP      OOO        OOO       PPP          PPP
PPPPPPPPPPPP          OOO        OOO       PPPPPPPPPPPP
PPPPPPPPPPPP          OOO        OOO       PPPPPPPPPPPP
PPPPPPPPPPPP          OOO        OOO       PPPPPPPPPPPP
PPP                   OOO        OOO       PPP
PPP                   OOO        OOO       PPP
PPP                   OOO        OOO       PPP
PPP                   OOO        OOO       PPP                    ......
PPP                   OOO        OOO       PPP                    ......
PPP                   OOO        OOO       PPP                    ......
PPP                      OOOOOOOOO          PPP                    ......
PPP                      OOOOOOOOO          PPP                    ......
PPP                      OOOOOOOOO          PPP                    ......
```

START Job LISTS1 Req #752 for O.ENABLEDOP Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>REXPL.POP.2, created: 3-Jun-86 21:03:39, printed: 11-Jun-86 11
Job parameters: Request created:11-Jun-86 10:53:57 Page limit:111 Forms:XERO
File parameters: Copy: 1 of 1 Spacing:SINGLE File format:ASCII Print mode:

```
FUNCTION WCAPEX XL;
NL(1);prstring('possible captures by the pawn are:!);nl(1);XL=>;nl(1);
end;

FUNCTION R1EX;
NL(1);PRSTRING('IF the pawns have captured all opposition pieces that!);
nl(1);PRSTRING('can be captured on board and these opposition pieces!);
nl(1);prstring('include a pawn that could not have reached a capture square!);
nl(1);prstring('then a pawn promoted.  [RULE001]!);NL(1);
END;

FUNCTION R2EX;
nl(1);prstring('If the pawns captured 8 pieces and an opposition pawn!);nl(1);
prstring('could not have reached a capture square then a pawn promoted.!);
nl(1);prstring('[RULE002]!);
end;

FUNCTION R3EX;
NL(1);prstring('If one side has > 2 bishops or > 2 rooks or >2 knights!);
nl(1);prstring('or >1 queen on the board then a pawn promoted.[RULE003]!);
end;

FUNCTION R4EX;
nl(1);prstring('If one side  has 2 bishops travelling on the same colour!);
nl(1);prstring('squares then a pawn promote.[RULE004]!);
end;

FUNCTION R5EX;
nl(1);prstring('If one side has a bishop captured on its home square and!);
nl(1);prstring('there is a bishop on board travelling on the same colour!);
nl(1);prstring('square then a pawn promoted.[RULE005]!);
end;

FUNCTION R6EX;
nl(1);prstring('if the pawns have made all their captures on one colour!);
nl(1);prstring('and among these captures is an opposition pawn that!);
nl(1);prstring('could not have reached a capture square!);
nl(1);prstring('then a pawn promoted.  [RULE006]!);nl(1);
end;

FUNCTION R7EX;
nl(1);prstring('if a pawn captured on promoting and the only piece it could!);
nl(1);prstring('have captured is an opposition pawn!);nl(1);
prstring('then an opposition pawn promoted  [RULE007]!);nl(1);
end;

FUNCTION R8EX;
nl(1);prstring('if a pawn made n captures in moving from its home square to!);
nl(1);prstring('a capture promotion square and the only pieces available!);
nl(1);prstring('for capture are n opposition pawns!);
nl(1);prstring('then an opposing pawn promoted  [RULE008] !);nl(1);
end;

FUNCTION R18EX XL;
nl(1);prstring('No captures permitted -  the following pawn !);nl(1);
prstring('reverse moves have been deleted!);nl(1);xl=>;
end;

FUNCTION R13EX XL;
nl(1);prstring('King must not be adjacent to king - following deleted!);
nl(1);XL=>;
end;
```

```
FUNCTION R14EX XL;
nl(1);prstring('Piece moved from check position - following deleted!);
nl(1);XL=>;
end;

FUNCTION R15EX XL;
nl(1);prstring('Own king in check and no way found for this to happen!);
nl(1);prstring('following deleted!);nl(1);XL=>;
end;

FUNCTION R16EX XL;
nl(1);prstring('Too many pawn captures - following deleted!);
nl(1);XL=>;
end;

FUNCTION R17EX XL;
nl(1);prstring('Number of pieces captured on board less than opposition!);
nl(1);prstring('pawn captures - following deleted!);nl(1);XL=>;
end;

FUNCTION R12EX XL;
nl(1);prstring('pieces given as not moving - following deleted!);
nl(1);XL=>;
end;

FUNCTION R19EX XL;
nl(1);prstring('A pawn revers moves to its home square and constrains!);
nl(1);prstring('a piece that was captured on board!);
nl(1);prstring('following deleted!);nl(1);XL=>;
end;

FUNCTION R30EX XL;
vars    PC P;
XL.HD.HD->PC;BORW(PC)->P;
nl(1);prstring('if the king or rook moved then!);sp(1);pr(P);sp(1);
prstring('cannot castle!);nl(1);
prstring('the following moves deleted!);nl(1);XL=>;nl(1);
end;

FUNCTION R31EX XL;
vars TLXL PC P;
XL.HD.HD->PC;XL.HD.TL->TLXL;BORW(PC)->P;
nl(1);prstring('a pawn promoted on!);sp(1);pr(TLXL);sp(1);
prstring('so the!);sp(1);pr(PC);sp(1);prstring('must have moved!);
nl(1);prstring('so!);sp(1);pr(P);sp(1);prstring('cannot castle on this side!);
nl(1);
end;

FUNCTION R32EX XL;
vars    HL Y X RK HHL PC KI;
BORW(XL.HD.HD)->PC;
If PC="W" then "WK"->KI;
else "BK"->KI;
close;
loop:   If XL.null then return;
else dest(XL)->XL->HL;HL.TL.HD->X;HL.HD->RK;HL.TL.TL.HD->Y;HL.TL->HHL;
nl(1);prstring('if the promoted!);sp(1);pr(RK);sp(1);
prstring('is on!);sp(1);pr(HHL);sp(1);prstring('then the!);
sp(1);pr(KI);sp(1);nl(1);prstring('must have moved to let it reach its square,so!);sp(1);
nl(1);          pr(PC); sp(1); prstring('cannot castle!);
goto loop;
close;
end;
```

```
FUNCTION R33EX XL;
vars    HL PC RK HHL;
BORW(XL.HD.HD)->PC;
loop:   If XL.null then return;
else dest(XL)->XL->HL;HL.HD->RK;HL.TL->HHL;
nl(1);prstring('if the promoted piece is the!);sp(1);pr(RK);
sp(1);prstring('on!);sp(1);pr(HHL);sp(1);prstring('then!);
sp(1);pr(PC);sp(1);prstring('cannot castle on this side!);nl(1);
goto loop;
close;
end;

FUNCTION R34EX XL;
vars    KI HL BW RK;
BORW(XL.HD)->BW;
If BW="B" then "BR"->RK;
else "WR"->RK;
close;
XL.HD->KI;XL.TL->HL;
nl(1);prstring('the!);sp(1);pr(KI);sp(1);prstring('on!);sp(1);pr(HL);
sp(1);prstring('moved to let out a!);sp(1);pr(RK);sp(1);
prstring('that was captured on board!);nl(1);prstring('so!);sp(1);
pr(BW);sp(1);prstring('cannot castle!);nl(1);
end;

FUNCTION R35EX XL;
vars HL P Y PWN KI;XL.HD->HL;HL.TL.HD->Y;
If Y=7 then "WP"->PWN;"BK"->KI;"B"->P;
else "BP"->PWN;"WK"->KI;"W"->P;
close;
nl(1);prstring('the promoting!);sp(1);pr(PWN);sp(1);
prstring('placed the!);sp(1);pr(KI);sp(1);prstring('in check from!);sp(1);
pr(HL);nl(1);prstring('so!);sp(1);pr(P);sp(1);prstring('cannot castle!);nl(1);
end;

FUNCTION R36EX XL;
vars PWN HL HHL;
loop:   If XL.null then return;
else XL.HD->HL;[%HL.TL.HD,HL.TL.TL.HD%]->HHL;
If HL.HD="WR" then "BP"->PWN;
else "WP"->PWN;
close;
nl(1);prstring('a!);sp(1);pr(PWN);sp(1);prstring('captured the original!);
sp(1);pr(HL.HD);sp(1);prstring('on!);sp(1);pr(HHL);nl(1);
TL(XL)->XL;goto loop;
close;
end;

FUNCTION R40EX XL;
VARS P KI HL PC XF YF XT YT;
BORW(XL.HD.HD)->P;
If P="B" then "WK"->KI;else "BK"->KI;close;
loop:   If XL.null then return;
else dest(XL)->XL->HL;HL.HD->PC;HL.TL.HD->XF;HL.TL.TL.HD->YF;
HL.TL.TL.TL.HD->XT;HL.TL.TL.TL.TL.HD->YT;
nl(1);pr(KI);sp(1);prstring('in check .!);sp(1);pr(PC);
sp(1);prstring('moved from!);sp(1);pr("(");pr(XF);pr(",");pr(YF);pr(")");
sp(1);prstrinG('to!);sp(1);pr("(");pr(XT);pr(",");pr(YT);pr(")");
nl(1);
goto loop;
close;
END;

FUNCTION R41EX XL;
```

```
nl(1);prstring('reverse move list!);nl(1);XL=>;
nl(1);
end;

FUNCTION R42EX XL;
nl(1);prstring('king in check and no way found for this to happen!);
nl(1);
end;

FUNCTION R43EX XL;
vars     XF YF XT YT;
nl(3);
XL.HD->P;XL.TL.HD->XT;XL.TL.TL.HD->YT;XL.TL.TL.TL.HD->XF;
XL.TL.TL.TL.TL.HD->YF;
nl(1);pr(P);sp(1);prstring('moved from!);sp(1);pr("(");
pr(XT);PR(",");PR(YT);PR(")");sp(1);prstring('to!);sp(1);
pr("(");pr(XF);pr(",");pr(YF);pr(")");
nl(1);
end;

FUNCTION R44EX XL;
nl(1);prstring('initial reverse move list!);nl(1);XL=>;nl(1);
end;

FUNCTION R45EX XL;
COMMENT used by WHPCAP;
vars    HL HHL;
nl(2);prstring('a king is in check.moving the checking piece leaves!);
nl(1);prstring('the same king still in check - implies capture!);
XL.HD->HL;
nl(2);prstring('possible captures are:-!);nl(1);HL=>;
XL.TL.HD->HL;
loop:   If HL.null then goto L1;
else dest(HL)->HL->HHL;
If HHL="BB" or HHL="WB" then nl(2);prstring('bishop not captured - wrong colour square!);
elseif HHL="BP" or HHL="WP" then nl(2);prstring('pawn not captured - could not reach captu
else nl(2);pr(HHL);sp(1);prstring('not captured,all moves to capture square place
                                                    opposition king in check!);
close;
goto loop;
close;
L1:     XL.TL.TL.HD->HL;
loop1:  If HL.null then return;
else dest(HL)->HL->HHL;
If HHL.HD.null then nl(2);prstring('checking piece could not have moved last - move
                                                    must have been!);nl(1); HHL.TL.HD=>;
exit;
nl(1);prstring('capture piece move!);nl(1);HHL.TL.HD=>;
nl(1);prstring('checking piece move!);nl(1);HHL.HD=>;
goto loop1;
close;
end;

FUNCTION R46EX XL;
COMMENT a king is in check - xl=variable chi;
vars    K P X Y;
XL.HD->K;XL.TL.HD->P;XL.TL.TL.HD->X;XL.TL.TL.TL.HD->Y;
nl(2);prstring('the!);sp(1);pr(K);sp(1);prstring('is in check from the!);
sp(1);pr(P);sp(1);prstring('on!);sp(1);pr("(");pr(X);PR(",");
pr(Y);pr(")");
end;

FUNCTION R47EX XL;
COMMENT a king is in check.xl gives possible ways it could have
                 happened;
nl(2);prstring('a king is in check.possible ways it could happen as follows!);nl(1);XL=>;
```

```
end;

FUNCTION R50EX XL;
vars    XA XC XS;
XL.HD->XA;XL.TL.HD->XC;XL.TL.TL.HD->XS;
nl(1);prstring('the pawn promoted on one of the following squares!);
nl(1);XA=>;
If not(XC.null) then nl(1);prstring('on promoting it captured on one of the
following squares!);nl(1);XC=>;
close;
nl(1);prstring('it crossed the 2nd/7th rank on one of the following squares!);
nl(1);XS=>;
nl(1);
end;

FUNCTION R52EX XL;
vars    HL PWN X;
XL.HD.HD->HL;HL.HD->PWN;HL.TL->X;
nl(1);prstring('the!);sp(1);pr(PWN);sp(1);prstring('from!);
sp(1);sp(1);pr(X);sp(1);prstring('promoted!);nl(1);
end;

FUNCTION R60EX XL;
nl(1);prstring('if 1 pawn captured on its 1st move!);nl(1);
prstring('and could only have captured an opposition pawn!);nl(1);
prstring('and the opposition pawn could not have made any captures!);nl(1);
prstring('and the home square of the capturing pawn is on tht opposition!);
nl(1);prstring('pawn promotion file!);nl(1);
prstring('then the pawn captured before the opposition pawn promoted [RULE024]!)
end;

FUNCTION R61EX XL;
nl(1);prstring('if a pawn captured on its 1st move!);nl(1);
prstring('and could only have captured an opposition pawn!);nl(1);
prstring('and the opposition pawn made n captures in promoting,where n is the!);
nl(1);prstring('number of pieces constrained by the capturing pawn on its!);
nl(1);prstring('home square!);nl(1);
prstring('then the pawn captured before the opposing pawn promoted [RULE 025]!);
NL(1);
end;


FUNCTION R64EX XL;
nl(1);prstring('the promoted piece is on board or was captured on one of the following
nl(1);XL=>;nl(1);                                                    squares!);
1->CAPBRD;
end;

FUNCTION R70EX XL;
nl(1);prstring('if there are 3 rooks or 3 knights or 3 bishops  or 2 queens!);
nl(1);prstring('on board!);nl(1);prstring('then one of them must be promoted!);
nl(1);prstring('the following is promoted!);nl(1);XL=>;nl(1);
end;

FUNCTION R71EX XL;
nl(1);prstring('if a side has more than 1 bishop travelling on the same colour!)
nl(1);prstring('then 1 must be promoted!);nl(1);
prstring('the following is promoted!);XL=>;nl(1);
end;

FUNCTION R72EX XL;
nl(1);prstring('if there is a bishop on board with its home square constrained!)
nl(1);prstring('by opposing pawns!);nl(1);
prstring('then it must be a promoted bishop [RULE028]!);nl(1);
```

```
prstring('the following is promoted!);nl(1);XL=>;nl(1);
end;

FUNCTION R73EX XL;
nl(1);prstring('if all possible promotion pieces are placed on the!);nl(1);
prstring('promoting squares!);nl(1);
prstring('then only the following can move!);nl(1);XL=>;nl(1);
end;

FUNCTION R74EX XL;
nl(1);prstring('the promoted piece is as follows!);nl(1);XL=>;nl(1);
end;

FUNCTION R75EX XL;
nl(1);prstring('if a promoting pawn captured on moving to the 2nd/7th rank!);
nl(1);prstring('it possibly captured on promoting.if the only pieces it !);
nl(1);prstring('could capture were 2 bishops!);
nl(1);prstring('then it could not have captured on promoting!);
nl(1);prstring('the pawn could not have promoted on the following squares!);
nl(1);XL=>;nl(1);
end;

FUNCTION R76EX HL;
nl(1);prstring('for each square on which the pawn may have promoted!);nl(1);
prstring('or for each piece it may have captured on promoting!);nl(1);
prstring('it has been deduced that an opposition pawn also promoted!);nl(1);
end;

FUNCTION R80EX XL;
vars M1 M2 P;
XL.HD->P;[%XL.TL.HD,XL.TL.TL.HD%]->M1;[%XL.TL.TL.TL.HD,XL.TL.TL.TL.TL.HD%]->M2;
nl(1);prstring('the king was placed in check by the!);sp(1);pr(P);sp(1);
nl(1);prstring('moving from!);sp(1);pr(M1);sp(1);prstring('to!);sp(1);
pr(M2);nl(1);
end;

FUNCTION R81EX XL;
nl(1);prstring('a pawn has therefore promoted!);nl(1);
end;

FUNCTION R82EX XL;
vars    HL KI PC;
loop:   If XL.null then return;
else dest(XL)->XL->HL;HL.HD->KI;HL.TL.HD->PC;nl(1);
prstring('the!);sp(1);pr(KI);sp(1);prstring('has been placed in check by the!);
sp(1);pr(PC);sp(1);prstring('on!);sp(1);pr(HL.TL.TL);nl(1);
goto loop;
close;
end;

FUNCTION R90EX XL;
nl(1);prstring('if all pawn captures occurred on one colour!);nl(1);
prstring('and these captures account for all missing pieces!);nl(1);
prstring('and these missing pieces include a bishop that could not have!);
nl(1);prstring('been captured on this colour!);nl(1);
prstring('then the missing piece must be a bishop  [RULE 090]!);nl(1);
end;

FUNCTION R100EX XL;
vars WL BL HL;
XL.HD->WL;XL.TL.HD->BL;
loop:   If not(WL.null) then WL.HD->HL;
nl(1);prstring('the wp on!);sp(1);pr(HL.HD);sp(1);prstring('captured a!);
sp(1);pr(HL.TL.HD);TL(WL)->WL;goto loop;
```

```
close;
nl(1);
loop1:  If not(BL.null) then BL.HD->HL;
nl(1);prstring('the bp on!);sp(1);pr(HL.HD);sp(1);prstring('captured a!);
sp(1);pr(HL.TL.HD);TL(BL)->BL;goto loop1;
close;
nl(1);
end;

FUNCTION R101EX XL;
vars HL HHL CL DL;
REV(XL)->XL;nl(1);prstring('the pawns captured in the following order:!);nl(2);
loop:   If not(XL.null) then XL.HD->HL;HL.HD->CL;HL.TL.HD->DL;
[%CL.TL.HD,CL.TL.TL.HD%]->HHL;
prstring('the!);sp(1);pr(CL.HD);sp(1);prstring('on!);sp(1);
pr(HHL);sp(1);prstring('captured a!);sp(1);pr(DL.HD);
if not(DL.TL.null) then sp(1);
prstring('from!);sp(1);pr([%DL.TL.HD,DL.TL.TL.HD%]);
close;
nl(1);TL(XL)->XL;goto loop;
close;
end;
```

```
RRRRRRRRRRR          UUU         UUU     LLL          EEEEEEEEEEEEEEE
RRRRRRRRRRRR         UUU         UUU     LLL          EEEEEEEEEEEEEEE
RRRRRRRRRRRR         UUU         UUU     LLL          EEEEEEEEEEEEEEE
RRR          RRR     UUU         UUU     LLL          EEE
RRR          RRR     UUU         UUU     LLL          EEE
RRR          RRR     UUU         UUU     LLL          EEE
RRR          RRR     UUU         UUU     LLL          EEE
RRR          RRR     UUU         UUU     LLL          EEE
RRR          RRR     UUU         UUU     LLL          EEE
RRRRRRRRRRR          UUU         UUU     LLL          EEEEEEEEEEEEE
RRRRRRRRRRRR         UUU         UUU     LLL          EEEEEEEEEEEEE
RRRRRRRRRRRR         UUU         UUU     LLL          EEEEEEEEEEEEE
RRR    RRR           UUU         UUU     LLL          EEE
RRR    RRR           UUU         UUU     LLL          EEE
RRR     RRR          UUU         UUU     LLL          EEE
RRR     RRR          UUU         UUU     LLL          EEE
RRR         RRR      UUUUUUUUUUUUUUU     LLLLLLLLLLLLLLL  EEEEEEEEEEEEEEE
RRR         RRR      UUUUUUUUUUUUUUU     LLLLLLLLLLLLLLL  EEEEEEEEEEEEEEE
RRR         RRR      UUUUUUUUUUUUUUU     LLLLLLLLLLLLLLL  EEEEEEEEEEEEEEE


PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPPPPPPPPPPP            OOOOOOOOO       PPPPPPPPPPPP
PPP         PPP       OOO       OOO     PPP         PPP
PPP         PPP       OOO       OOO     PPP         PPP
PPP         PPP       OOO       OOO     PPP         PPP
PPP         PPP       OOO       OOO     PPP         PPP
PPPPPPPPPPPP          OOO       OOO     PPPPPPPPPPPP
PPPPPPPPPPPP          OOO       OOO     PPPPPPPPPPPP
PPPPPPPPPPPP          OOO       OOO     PPPPPPPPPPPP
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......
PPP                   OOO       OOO     PPP                    ......


*START* Job LISTS1 Req #752 for O.ENABLEDOP    Date 11-Jun-86 11:10:05 Monitor:
File RS:<R.ALDEN>RULES1.POP.2, created:  3-Jun-86 20:01:19, printed: 11-Jun-86 1
Job parameters: Request created:11-Jun-86 10:53:57   Page limit:111   Forms:XERO
File parameters: Copy: 1 of 1   Spacing:SINGLE   File format:ASCII   Print mode:
```

```
FUNCTION START;
.setupdb;
end;
function setupdb;
comment  SETS UP THE STATIC DATABASE, I.E. CONVERTS THE TRIPLES IN SDB.POP
         INTO RECORDS OF TYPES "PARAM", "CTYP" AND "RULE", CONVERTS THE TABLES
         IN TABLES.POP INTO RECORDS OF A CLASS SPECIFIED AT THE HEAD OF EACH
         TABLE, AND CONVERTS LISTS.POP INTO NAMED LISTS;
vars RG;
recordfns("rule",[compnd compnd compnd])->explain->premise->
category->destrule->consrule;
comment  A DUMMY RECORD "RG" IS SET UP;
consrule(nil,nil,nil)->RG;
.conslist;
.consdata;
end;
```

```
function conslist;
comment  CONVERTS LISTS.POP INTO LISTS;
vars lisread dataread A Y Q;
popmess([in LISTS1.POP])->lisread;
incharitem(lisread)->dataread;
loopif (.dataread->A;a/=termin)
       then .dataread->q;
       .buildl->valof(A);
close;
copy(ALLRULES)->Y;
loopif Y /= nil
       then copy(RG)->valof(hd(Y));
            tl(Y)->Y;
close;
end;


function consdata;
comment  CONVERTS SDB.POP INTO RECORDS;
vars sdbread dataread A B C;
popmess([in RULES1.POP])->sdbread;
incharitem(sdbread)->dataread;
comment  SDB.POP IS IN THE FORM OF OBJECT - ATTRIBUTE - VALUE TRIPLES, WHERE
         THE OBJECT IS A PARAMETER, CONTEXT TYPE OR RULE AND IS A WORD, THE
         ATTRIBUTE IS THE NAME OF A PROPERTY AND IS A WORD AND THE VALUE IS
         THE VALUE OF THAT PROPERTY AND CAN BE A WORD, LIST, STRING OR ANY
         OTHER DATA TYPE. THE OBJECTS ARE ALL NAMES OF RECORDS, THE PROPERTIES
         ARE THE NAMES OF THE FIELDS OF THESE RECORDS AND THE VALUES ARE THE
         VALUES OF THOSE FIELDS;
comment  THE FIRST (NEXT) ITEM TO BE READ (ASSIGNED TO THE VARIABLE A) IS
         EITHER THE NAME OF A RECORD OR "TERMIN";
loopif (.dataread->A; A /= termin)
       then comment  THE NEXT TWO ITEMS TO BE READ ARE A PROPERTY AND ITS
                     VALUE. THESE ARE ASSIGNED TO THE VARIABLES B AND C
                     RESPECTIVELY;
            .dataread-> B; .dataread->C;
            comment  IF THE VALUE IS A LIST THEN C WILL BE "[", BUT WE NEED
                     C TO BE THE ENTIRE LIST;
            if C = "[" then buildl()->C; close;
            comment  THE VALUE OF THE COMPONENT IS UPDATED;
            C->valof(B)(valof(A));
close;
end;
```

```
function buildl=>LL;
comment  READS A LIST FROM A PREVIOUSLY SPECIFIED FILE;
vars D;
nil->LL;
loopif (.dataread->D; D /= "]")
       then if D = "["
               then buildl()->D;
            close;
            LL<>[%D%]->LL;
close;
end;
function DAND LIST => ED;
if TRACE = 1 then pr('DAND !);pr(LIST);nl(1);close;
comment  THIS IS A TOP-LEVEL FUNCTION WHICH IS CALLED BY THE PREMISE OF ALL
         RULES. IT FINDS THE MINIMUM CF RETURNED BY THE CLAUSES FOR USE IN
         THE ACTION PART OF THE RULE. RETURNS 0 IF ANY CLAUSE FAILS;
vars T CLAUSE CLAUSELIST;
copy(LIST)->CLAUSELIST;
loopif CLAUSELIST /= nil
       then hd(CLAUSELIST)->CLAUSE;
            tl(CLAUSELIST)->CLAUSELIST;
if TRACE = 1 then pr('CLAUSE=!);pr(CLAUSE);nl(1);close;
            comment  THE FIRST CLAUSE IS EVALUATED. IF IT RETURNS FALSE THEN
                     DAND RETURNS FALSE;
            if not(eval(CLAUSE))
               then 0->ED; return;
                           if TRACE = 1 then pr('T=!);pr(T);nl(1);close;
            close;
close;
1->ED;
end;

function eval LIST;
if TRACE = 1 then pr('eval!);sp(1);pr(LIST);nl(1);close;
comment  ENABLES A PREMISE CLAUSE TO BE INPUT AS A LIST, WHERE THE FIRST
         ELEMENT IN THE LIST IS THE FUNCTION TO BE EVALUATED, AND THE OTHER
         ELEMENTS ARE ITS ARGUMENTS;
vars FN X;
comment  THE FIRST ELEMENT IN THE LIST (A FUNCTION) IS ASSIGNED TO THE
         VARIABLE FN;
valof(hd(LIST))->FN;
tl(LIST)->LIST;
loopif LIST /= nil
               then valof(hd(LIST));
                           tl(LIST)->LIST;
close;
comment  THE FUNCTION FN IS APPLIED USING ITEMS FROM THE STACK AS ITS
         ARGUMENTS;
apply(FN);
end;
```