How to read, make and store chess moves

A. G. Bell* and N. Jacob†

A vast wealth of knowledge lies untapped in the chess libraries of the world. This paper discusses some simple techniques which could allow computers to access, verify and share this information.
(Received October 1977)

1. Introduction
Chess games have been recorded for many centuries, but the records are not easily accessed or shared by computers. One reason is that the notations used are 'natural', i.e. they were evolved by humans for humans and consequently are untidy, complex, redundant and prone to human error.

This paper describes how a program can read English and algebraic game records by the use of 'skeleton keys'. In order to exemplify the technique (and also reduce the description of trivia to a minimum) a working program is given in the Appendix; the program is in PASCAL (Jensen and Wirth, 1974), a language similar to ALGOL, and should be used more as a footnote to the paper than as an example of efficiency. The paper also proposes an international machine notation which could, in the future, allow computers to share more easily the processed game records.

2. English notation
Essentially this is a descriptive shorthand for MY MAN (M1) IN SQUARE (S1) TAKES OPPONENTS MAN (M2) IN SQUARE (S2) or S1(M1) → S2(M2) where M:: = (empty), P (pawn), N (knight), B (bishop), R (rook), Q (queen), K (king) and S is a (two letter, one digit) square description as shown in Fig. 1(a).

The common opening move P-K4 is, therefore, a skeleton form of

\[ \text{KK2(P)} - \text{KK4( )}. \]

In order to identify this move the program builds up the skeleton in the character variables P1, P2, P3, P4, P5, P6, P7, P8 thus

<table>
<thead>
<tr>
<th>S1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>P</td>
<td>K</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>K</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

It then compares all the non '·' characters in this skeleton with all the possible pawn moves and finds that only

<table>
<thead>
<tr>
<th>S1</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
<th>P5</th>
<th>P6</th>
<th>P7</th>
<th>P8</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>K</td>
<td>2</td>
<td>P</td>
<td>K</td>
<td>K</td>
<td>4</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

gives full agreement and the corresponding white move is made, i.e. square 13 to square 29 in the program's board notation. (See Fig. 1(b)).

There are four special cases which the skeleton cannot handle. The first is the en passant capture (P*PE)—in this case the program shifts the opponent's pawn back one square and then lets nature take its course (see the Appendix for exact details of this case and the two following).

The second special case is king or queen side casting (0-0 or 0-0-0). These moves are flagged by making the piece to move have an apparent value of 9. On returning from SKELETON the program checks for this flag and, if true, makes the correct rook move and primes the king move.

The third special case is pawn promotion, e.g. P-KQ0. In this case, the program remembers the promotion character 'Q' and, on completion of the move, replaces the pawn with the correct piece.

The fourth special case is when the uniqueness of a move depends on whether or not it is a check/checkmate move or on the fact that similar moves are illegal. The program does not handle such problems. To illustrate the possible complications, consider Fig. 2. Black has just played P/2-Q4 and the program, faced with this position and the input P*P, would find an eight-way ambiguity (an octigity?). Note that a human would also find P*P to be an octigity but his eight possibilities are not the same as the program's because the human includes the two en passant captures and excludes the two illegal captures whereas the program does the opposite.

Fig. 2 also gives a minimal unambiguous notation for each of the eight possible moves plus the skeletons the program would produce plus the correct move in full. Now an experienced human can identify and isolate each of the moves correctly because

(a) he knows what en passant really means
(b) he can use the presence or absence of check (+) or checkmate (+) and
(c) he knows that the pawn in KN2 cannot make captures.

But the simpleminded program has none of this knowledge and so it still cannot uniquely identify any of the moves from their skeletons. To illustrate an even greater complexity the reader is invited to identify the move P*R++; a move which requires even greater sophistication to identify correctly.

At this point the reader might be puzzled as to why we take such pains to reveal the shortcomings of the program. The important point to appreciate is that the program has no real understanding of what it is doing—it can, for example, quite happily obey the move Q*K. It is therefore necessary to appreciate how successfully this simple program can perform.

As test data for the program 1000 moves were copied out of the Ruy Lopez section of the Encyclopaedia of Chess Openings, Vol. C ECO 90-99. These moves are recorded in algebraic notation (see next section) and, when processed, the program was able to isolate and make 973 moves correctly. Of the remaining 27 moves, 17 were genuinely ambiguous in the original text, six were erroneous and only four were ambiguous.

*Division of Computing Research, CSIRO Canberra, Australia; now at Computing Services, University of Sheffield, The Hicks Building, Sheffield S10 2TN.
†Department of Economics, University of Newcastle, Newcastle, N.S.W., Australia.
in the program's terms, i.e. could have been correctly resolved
by analysis of the check and illegality information. It therefore
seems ridiculous to double (at least) the size of the program
to handle 4/1000 = 0.4% of the data correctly, particularly
when the majority of problems were produced by incorrect
data in the original text or caused by faulty transcription.

The interesting feature of the 17 genuine ambiguities in the
sample is that they are almost all caused by confusion of rooks,
for example, in Fig. 2, if white's move is R-Q7. Humans can
often resolve these ambiguities by looking at the moves that
follow, e.g. if black's reply to R-Q7 is R*R, then we know that
it must have been the rook in QB7 that had just moved.

The program also (very crudely) allows what follows to resolve
any current ambiguity by simply flagging (it outputs '??') and
then making one of the possible moves. Should it eventually
find an impossible move it halts and a human scan of the out-
put, with particular attention given to the flagged ambiguities,
can usually resolve the problem—an example of ambiguity is
given in Fig. 4.

3. Algebraic notation
Essentially this is a shorthand for MY MAN (M1) IN
SQUARE (S1) GOES TO SQUARE (S2) or (M1)S1 → S2,
where M is the same as in English notation and S is a (one
lower case letter, one digit) square description as shown in
Fig. 1(c).

Note that in this notation (a) pawn moves are simply the
square a pawn can move to, e.g. the opening move P-K4 is
e4 not Pe4 and (b) captures are not distinguished from ordinary
moves unless they are pawn captures in which case the file of
the pawn is given as identification (Comparison of the English
and algebraic examples in the Appendix should make these
conventions clear).

In order to identify the correct move the program builds up an algebraic skeleton in the character variables P1, P2, P3, P4, P5 thus

\[
\begin{array}{c|c|c|c|c}
(M1) & S1 \rightarrow S2 \\
\hline
P1 & P2 & P3 & P4 & P5 \\
\hline
P & E & E & E & 4 \\
\end{array}
\]

It then compares all the non-\(\ldots\) characters in the skeleton with all the possible pawn moves and finds that only

\[
\begin{array}{c|c|c|c|c}
(M1) & S1 \rightarrow S2 \\
\hline
P & P & E & 2 & 4 \\
\end{array}
\]

gives full agreement, i.e. square 13 to square 29.

Algebraic notation (if written accurately) is much more suitable to the program than English notation; in fact, the program can resolve all the pawn captures in Fig. 2. Unfortunately the notation does depend on using lower case letters for squares and upper case letters for pieces. Because the program cannot distinguish upper and lower case letters it is possible to confuse a bishop capture with a b-file pawn capture, e.g. in Fig. 2, if there was a white bishop on square b4, the program would flag an ambiguity and assume that BA3 is more likely to be Ba3 than Ba3. Note that (a) the correct move can be forced by entering it as PB2A3 and (b) none of the other pieces can cause this problem because their initial letters in PASCAL, are all greater than ‘H’.

One would imagine that algebraic notation is preferable to English notation but, in terms of error detection and correction, this is not true. The confusion of rooks can still occur and is much more difficult to resolve because capture is not indicated; e.g. in Fig. 2, if white plays RD7 then black’s reply RD8 does not resolve the ambiguity as it did in the more redundant English notation of R*R.

Even worse is that human error resulting in impossible moves is much more likely in algebraic records—typically the file is out by one letter and the rank by one digit. To recover from such errors requires an extremely knowledgeable scan of what follows and, although a human can often correct such errors, we have no idea how a program might reproduce this ability.

4. International notation

In order to ease the exchange of processed chess games between computers we propose the following data base standard for chess pieces, board and move notation.

1. The pieces are K = 6, Q = 5, R = 4, B = 3, N = 2, P = 1.

2. The board notation is given in Fig. 1(d) and is effectively the program’s decimal notation less 1 and written in octal.

3. The move notation derives from the piece and board notation. First note that all possible moves can be defined by
four octal digits with the following provisos:

(a) if the piece is a pawn moving diagonally to an empty square then it must be an en passant capture

(b) if the piece is a king moving further than one square then it must be casting

(c) if the piece is a pawn moving into a promotion square then the rank of the final square denotes the promotion, e.g. in Fig. 2 6655 means P*RQ, 6645 means P*RR, 6635 means P*RB and 6626 means P*N8N.

Now, although it is possible to compress all moves into four octal digits, this is not good practice partly because this superpacked form requires a fairly elaborate program to decipher it and partly because most computers operate on 8-bit bytes rather than 6 bits.

We therefore propose that a complete move be defined by 16 bits, thus (F) (S1) (P) (S2), where F (Flag) is 2 bits, S1 (from) is 6 bits, P (promotion) is 2 bits and S2 (to) is 6 bits. The deciphering algorithm is then much simpler (note that ( ) means contents of ).

(S2) := (S1); (S1) := 0;
if (F) = 1 then (* PROMOTION *)
(S2) := (S2) + (P) + 1;
if (F) = 2 then (* EN PASSANT *)
if S2 > S1 then (S2 - 8) := 0 else (S2 + 8) := 0;
if (F) = 3 then (* CASTLING *)
if S2 > S1 then
begin
(S2 - 1) := (S2 + 1); (S2 + 1) := 0 end else
begin
(S2 + 1) := (S2 - 2); (S2 - 2) := 0 end

5. Program description

The program has the structure
MAIN PROGRAM (LISTMOVES(SKELETON, COMPARE, READMOVE, NORKM, RORBMP)).

In order to understand how NORKM (Knight or king move), RORBMP (rook or bishop move) and WORBPM (white or black pawn move) work the reader must consult Algorithm 50 (Bell, 1970).

The tables driving this program are the same as those for Algorithm 50 and the first operation, on entry, is to read them into the arrays KN (Knight), KNG (king), ROOK (rook), BSHP (bishop), WP (White pawn) and BP (Black pawn).

The program then clears the board and reads in the positions. For the opening position the data is

```
4 1 2 3 4 5 3 6 2 7 8
1 9 1 1 1 1 1 1 1 1 1
1 1 2 1 1 1 1 1 1 1 1
1 4 1 1 1 1 1 1 1 1 1
4 5 7 5 3 5 6 6 6 6 6
```

The program then reads the board notation for each of the 64 squares into the arrays D1, D2, D3, D4. English notation is:

```
QR18QN18QB18Q18KK18KB18K18KB18K18KR18
QR27QN27QB27Q27KK27KB27K27KR27
QR36QN36QB36Q36KK36KB36K36KR36
QR45QN45QB45Q45KK45KB45K45KR45
QR54QN54QB54Q54KK54KB54K54KR54
QR63QN63QB63Q63KK63KB63K63KR63
QR72QN72QB72Q72KK72KB72K72KR72
QR81KN81KB81Q81KK81KB81K81KR81
```

The program sets the en passant aid (BACK := 8), the line counter (LINE := 1) and then begins to process the white and black moves (LISTMOVES (WMEN,BMEN,D3); LISTMOVES (BMEN,WMEN,D4)).

In order to make the program accept algebraic input the procedures SKELETON, COMPARE and READMOVE have to be replaced in the program and the algebraic board notation replaces the English in the data thus:

```
A1...B1...C1...D1...E1...F1...G1...H1...
A2...B2...C2...D2...E2...F2...G2...H2...
A8...B8...C8...D8...E8...F8...G8...H8...
```

In order to demonstrate the program the following game was entered in English and algebraic.

<table>
<thead>
<tr>
<th>English</th>
<th>Algebraic</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-K4</td>
<td>P-Q4</td>
</tr>
<tr>
<td>P-K5</td>
<td>P-K4</td>
</tr>
<tr>
<td>P*PE</td>
<td>Q*Q3</td>
</tr>
<tr>
<td>P*KP</td>
<td>B-Q2</td>
</tr>
<tr>
<td>P*BB</td>
<td>Q*B</td>
</tr>
<tr>
<td>P-QN4</td>
<td>P-Q5</td>
</tr>
<tr>
<td>P-N5</td>
<td>P-B4</td>
</tr>
<tr>
<td>P*PE</td>
<td>Q*P</td>
</tr>
<tr>
<td>P-B4</td>
<td>P*PE</td>
</tr>
<tr>
<td>B-B4</td>
<td>B-BP</td>
</tr>
<tr>
<td>N*P</td>
<td>N-B3</td>
</tr>
<tr>
<td>N-R3</td>
<td>N-R3</td>
</tr>
<tr>
<td>N-K2</td>
<td>0-0-0</td>
</tr>
<tr>
<td>N-3-B4</td>
<td>R/R-B1</td>
</tr>
<tr>
<td>0-0</td>
<td>Q*K</td>
</tr>
</tbody>
</table>

The outputs of both versions of the program are given in Figs. 3 and 4. Note that the English output gives the moves in fully compressed International whereas the algebraic output is given in the program’s decimal board notation.

One final comment. English notation does not have any fixed standard for its symbols and the characters ( ) , | and = are often used. When preparing data for the program given here, these symbols, together with check (+) and checkmate (+ +), should be omitted whilst any en passant indication (usually e.p.) should be represented by just the letter ‘E’. Even better would be to modify the procedure READ so that it ignored the characters ( ), , | = and +, treated ( as / and, on reading the letter ‘E’, threw away all characters until a space was read. The program should then be able to read and verify almost any chess game in English notation with considerably more accuracy than any human.

6. Conclusion

The purpose of this paper is to present the reader with guidelines and illustrations as to how to process, verify and store chess game records. In order to make clear how the skeleton technique works a great deal of the detail has not been discussed. The reader must therefore consult and understand the program given in the Appendix for clarification on a number of trivial problems. He will, if sufficiently perspicacious, find a few errors—for example, the program does not perform correctly for P*BN, i.e. pawn takes a bishop and is promoted to a knight. Our only defence, as already mentioned, is that errors due to the program will be considerably less than errors due to data. It is hoped that, within a decade, programs following the techniques described here will have processed and stored a great deal of the chess games recorded and thus make available to computers a great deal of the knowledge which, up till now, has only been available to literate humans.

Acknowledgements

The authors wish to thank D. Slate, L. Atkin, M. Donsky, J. Birmingham and P. Kent for their comments on the proposed International Notation.
Appendix Pascal program to read English notation

```pascal
PROCEDURE SKELUT; 
VAR X: INTEGER; 
CONST X = 0; 
BEGIN 
  X := 0; 
END;
```

References

The Computer Journal Volume 22 Number 1 75