5.5 Robot Chess

D. Prinz

First published in *Research,* 1952, pp.261-266. Reprinted by permission of Butterworth and Co. (Publishers) Ltd.

In recent times digital computers have been increasingly used as aids in the study of scientific and economic problems. Dr. Prinz describes how such a machine can be 'programmed' to solve simple chess problems successfully, though it does so more slowly than a human chess player.

The age old dream of machines playing games of strategy, like draughts or chess – and playing them well enough to beat human opponents – has recently been revived for two reasons. One is the realization that problems of economics can be analyzed by means of logical structures similar to those implied in games of strategy. J. VON NEUMANN and O. MORGENSTERN have recently discussed this approach in great detail (1947). This seems to lift the interest in game-playing machines out of the realm of mere entertainment into the class of respectable scientific research; if machines can be made to play games, they should be able to solve economic problems of similar structure.

The second reason is the advent of the 'electronic brains' or, to give them their proper name, universal high speed electronic digital computers. The emphasis is on the 'universal'. The same machine can be made to solve a problem in aircraft design in the morning, compute optical lens systems in the afternoon and work out PAYE wage lists in the evening. You can make it do all this without any alterations in its design, without plugging in new connections, simply by providing it with suitable information, usually contained on a punched tele-printer tape or similar device. The problem is no longer 'making a machine to play chess' but rather 'making a machine play chess'. You do not have to be an inventor or engineer; all you need is paper, pencil and patience. You also need access to an existing electronic computer, including the apparatus for converting your notes into a punched paper tape, and a knowledge of the 'code' used by the machine *i.e.* the system for writing down information in such a form that the machine may understand and carry out your instructions.

The complete list of instructions (or 'orders' or 'commands') required for any particular problem is called a 'programme', and the technique of designing programmes, or of 'teaching' the machine what to do, is known as 'programming'. Subsequences of instructions in a programme are often called 'routines'.

The public interest in robot chess is reflected in the recent literature: I mention only two papers which have appeared in the *Science News* series and which deal with the theoretical aspects of the problem (Davies, 1950) and describe special machines built in the past (Byard): they may be consulted for further references. In addition, the daily press has taken notice. In a report to the *Manchester Guardian* (13th November, 1951) ALISTAIR COOKE has mentioned an American firm which was said to have challenged any human chess player to a match against its electronic machine. However, inquiries in the United States have so far failed to confirm this report and it seems that the first concrete approach to the problem is the work carried out on the machine* installed at the University of Manchester. This work will be briefly described here.

The problem

At the outset it must be made clear that the aim of this experiment was rather modest. No attempt was made to let the machine play a successful game of chess. What has been done is a demonstration of the ability of the machine to solve simple chess problems.

*Built by Messrs Ferranti Ltd, Manchester.

A chess problem, as every player knows, is the problem of finding a move, the 'key move', which will mate the opponent, irrespective of his replies, in a given number of further moves. Only the simplest case, the 'mate in two', has been treated, and even so, some of the moves permitted by the rules of chess (double pawn moves, castling and the exchange of a pawn for a piece on reaching the last row) have been excluded to make the programme as simple as possible. Finally, the possibility of a stale-mate was admitted as a solution.

To anticipate the result of the experiment, it has turned out that the machine, when provided with the programme to be described, can make chess moves and find the solution of chess problems within the limits stated above; but it does so much more slowly than a human chess player.

One reason is that the machine is built to carry out arithmetic operations, and the moves on the chess board have to be broken down into a series of such operations before the machine can deal with them. If you are told that the white King is in one corner of the chess board, and that the black King is in another corner, the remainder of the board being empty, you know at once that neither King is in check. But the machine is in the position of a man who has to carry out an arithmetic operation to find which square to examine next and, having arrived there, finds instead of a chessman a piece of paper telling him which arithmetic operations this chessman is allowed to perform in order to reach other squares. Altogether, several hundred operations may be needed to find out whether or not the King is in check, and even at electronic speeds the whole process may take an appreciable fraction of a second.

Another reason is that the present programme does not eliminate unlikely moves: the machine is forced to investigate every possible move, until the solution is found. Even for a two-mover, the total number may run into several thousand on the average, and will be below a thousand only for particularly simple positions. Since every move has to be followed by a test for legality, it may consume something like 1 or 2 seconds, and the total time required for the solution may be several hours. The extremely simple problem illustrated in *Figure 1* took about 15 minutes to solve: most chess players could find the solution in less time than this.

Since the rules of chess usually impose a time limit, the machine would stand no chance of winning a game against a human player; but it could (with minor modifications to the programme) be made to play a game against another machine, or against itself. The possibility must, of course, not be excluded that by refined programming, by the use of faster machine, or both, a superior game played by a universal computer may finally be realized, not to mention the possibilities of special machines built for the purpose of playing chess only.

The Machine

Before going into a detailed description of the programme, it will be necessary to state briefly the essential features of the electronic computer.

The nucleus of the machine is a store in which patterns, consisting of two different units (like the dots and dashes of the Morse code), can be kept. These patterns can be interpreted as numbers, letters or other pieces of information. The exact correspondence does not matter here. The non-mathematical reader may think of the Morse code the teleprinter code, or the Braille script. For the mathematical reader it may be sufficient to hint at the binary number system, a form of digital representation in which the only digits used are 0 and 1. Neither is the actual form of storage important at this point. Suffice it to mention that the Manchester machine employs two forms of storage, one using magnetized elements on the surface of a rotating drum (similar in principle to magnetic tape or wire recording), the other using electrical charges on the screens of cathode ray tubes, similar to those contained in television receivers. The punched tape containing the original information supplied to the machine may be regarded as an additional, external store.

In the cathode ray or 'electronic' store the information is arranged in successive lines, each line containing either a number or an instruction. The lines themselves are numbered consecutively and the number of a line is often called its 'address'. This address or number of a line must be distinguished from the number contained in the line.

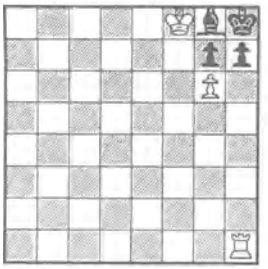

Figure 1

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

Figure 2

90	91	92	93	94	95	96	97	98	99
80	81	82	83	84	85	86	87	88	89
70	71	72	73	74	75	76	77	78	79
60	61	62	63	64	65	66	67	68	69
50	51	52	53	54	55	56	57	58	59
40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

Figure 3

An instruction consists of two parts, the address and the function part. When the machine 'reads' an instruction, it rearranges its internal connections just as an automatic telephone exchange connects you to another subscriber when you supply information by dialling a telephone number. More specifically the machine will be 'connected' to the line the address of which forms the address part of the instruction, and it will then either carry out an operation on the number contained in this line, or send the result of a previous operation to this line. Thus, an instruction of the form '53,A' may be interpreted to mean 'add the number standing in line No. 53 into the addition register', and the instruction '54,B' may mean 'send the contents of the addition register to line No. 54'. The time required to read and carry out an instruction is different for different computers: for the Manchester machine it is about one thousandth of a second.

Under the influence of a 'control unit' the machine goes normally through the instructions in their natural sequence and obeys them one by one. However, amongst these instructions there are some which in turn react on the control unit: their effect is to interrupt the normal sequence, causing the machine to 'jump' to some other instruction in the store. In particular, it is possible to go back to a previous instruction so that the same cycle of instructions is repeated over and over again. By making these 'control transfer instructons' dependent on certain arithmetic conditions it is possible to leave this 'loop' of repeated cycles after a specified number of times, or when a specified result has been obtained.

Since instructions are kept in the store together with numbers, they can themselves become the object of arithmetic operations. In particular, it is possible to modify their address part. Thus, by adding 1 to the address part, the instruction '53,A' can be altered to '54,A'. This facility is quite fundamental for the applicability of machines of this type. Suppose we wanted to add together the numbers in lines 100, 101, 102 ... up to, say, 199. If we had to write down all the instructions 100, A, 101, A ... 199, A explicitly and punch them out, there would not be much point in having a machine obeying these instructions at a rate of a thousand per second. Instead, we only write '100,A', followed by instructions adding 1 into the address part, and a control transfer instruction leading back to the beginning and so conditioned that the machine stops (or goes to some other sequence) when '199,A' is reached.

Now consider another case. Suppose we have a table of numbers stored in, say, lines No. 200, 201, ... 249. Assume that the machine has computed a number x within this range and put it into line 50. We wish to carry out an operation (symbolized, say, by the letter C) on the number standing in line No. x of our table. All we need is an instruction '0,C' combined with an addition of the contents of line No. 50 to the address part of this instruction. The resulting instruction 'x,C' will do what we want.

What this means is that the machine can 'look up' a table or directory just as we can find the logarithm of a given number from a mathematical table, or the telephone number of a given person in the directory. Expressed mathematically, the machine can find, and operate on, the values of a function $f(x)$ of an integral variable x provided the function f is tabulated in the store in such a way that line No. x contains the

function value f(x).

It should be mentioned here that the Manchester machine has a special facility for curtailing this process of modifying instructions. By making use of this facility, or by using conditional control transfer instructions, it is possible to make the machine go to different subsequences according to the number contained in a specified storage line. Thus, by allocating different numbers to the six chessmen (*e.g.* King = 2, Knight = 3), the machine can be made to enter a sequence dealing with King's moves when a certain storage line contains 2, and to enter a sequence dealing with Knight's moves when this line contains 3.

The Programme

It will now be possible to describe the essential features of the programme by means of lists or tables, consisting of pieces of information contained in successive store lines. The machine goes through these lists under the control of 'counters', certain numbers (kept in other storage lines) which are increased by 1 or 2 at appropriate stages in the programme.

To describe the first of these lists, imagine all the thirty two pieces* numbered consecutively, *e.g.* from 0 to 7 for the eight white Pawns: 8, 9 for the white Knights; 10, 11 for the white Bishops; 12, 13 for the white Rooks; 14, 15 for the white King and Queen; the black pieces are similarly numbered from 16 to 31. The number thus allocated to each piece is called the 'piece counter' and symbolized by j_p. It must be distinguished from the 'piece number' p which characterizes the six different pieces irrespective of colour, for instance, as follows:

Pawn	King	Knight	Queen	Bishop	Rook
1	2	3	4	5	6

We can now describe any particular position by means of a list consisting of 32 consecutive lines which we will number a, a+1, a+2, ... a+31, and which are so allocated to the 32 pieces, that line $a+j_p$ corresponds to the piece with the piece counter j_p. Each of these lines contains the number s of the square on which the piece is to be found: if the piece is not on the board at all the line will contain some special symbol, *e.g.* a negative number, indicating this fact. This list will be referred to as the 'a-table'.

An additional list, the 'b-table', describes a position in a different way. It allocates successive lines, say, b, b+1, b+2 ... to the squares on the board so that line $b+s$ coresponds to square No. s and contains information regarding the piece occupying this square, such as its piece counter j_p, its piece number p and a special symbol indicating its colour. A further symbol is used if the square in question is empty.

Moves from one square s to another square t are made by adding certain numbers to the original square number s to obtain t. For instance, we may allocate numbers from 0 to 63 to the 64 squares as indicated in *Figure 2*; then a move one step to the right will be effected by the addition of 1, one step upwards by the addition of 8 *etc.*

However, this scheme involves a particular difficulty when we come to the edge of the board. When the machine has, say, reached square 7 and tries a step to the right, it will compute the new square as 8 which is obviously wrong. We must, therefore, devise a scheme which will tell the machine when the edge of the board is reached. The solution is indicated in *Figure 3* which shows a 10×10 board in which the squares are numbered from 0 to 99. Of these 100 squares, only the central 64 constitute the proper or active board while the remaining 36 form a border. If s is the number of a border square then line $b+s$ will contain a symbol informing the machine that the limits of the active board have been exceeded.

A single-step move to the right is still carried out by the addition of 1, but a single step upwards now means addition of 10. To move diagonally upwards and to the right, we have to add 11. In general, there is a direction number d associated with each of the possible 8 directions. Counting these directions by a direction counter j_d, we can put successive numbers of d into successive lines $c+j_d$ of a list starting with line c. Beginning with moves to the right and proceeding in an anti-clockwise sense, we thus get the following 'c-table' or 'direction table':

*In correct chess parlance Pawns are not included among the 'pieces'. This distinction will not be made here.

j_d	*Line No.*	d
0	c	+ 1
1	$c+1$	+11
2	$c+2$	+10
3	$c+3$	+ 9
4	$c+4$	− 1
5	$c+5$	−11
6	$c+6$	−10
7	$c+7$	− 9

The 'step counter' j_s counts the number of steps of a move in a given direction. A move of j_s steps in the direction d is produced by the addition of the product $d \times j_s$ to the original square number s. For the King and, according to the restrictions imposed on the programme, for Pawns as well, j_s can only have the value 1.

Knight's moves are composed of horizontal or vertical single-step moves, followed by a diagonal single-step move. In principle, the same direction table can be used, but in practice it has been found more convenient to have a seperate list.

In addition to j_p, j_d *and* j_s, there is a colour counter j_c which decides whether the machine is making a white or black move. The values of j_c have been chosen as 0 for white, 16 for black. This choice is related to the fact that corresponding white and black pieces are 16 lines apart in the a-table. Instructions modified by j_c will select a piece of the correct colour from this list.

In what follows it will be convenient to talk about a first and second, white and black turn, during each of which a number of moves are tried out. With this nomenclature, White tries out all its first-turn moves until a solution is found. Black answers each of these moves with his first-turn moves until one is found for which no white second-turn move leads to a mate, and so on. A more detailed analysis of the structure characterizing the mate-in-two problem is given at the end of this article.

Let us now consider the case that White has made a number of first-turn moves without finding a solution. How does the machine construct the next first-turn move? White's last first-turn move was characterized by certain values of s, p, j_p, j_d and j_s (j_c being zero) and the machine has stored these numbers, together with the a-table and the b-table describing the original position, in the magnetic store in order to free the electronic store for the information needed in the moves tried out during the other turns. At the beginning of the new moves, all these data are transferred back into the electronic store.

The machine then starts by looking up the line containing the piece number p. Under the control of this number, it will then enter a sequence related to the piece having this number. If, for instance, p=5, it will enter the sequence for Bishops moves: they are characterized by odd direction counters. Next, the step counter j_s is increased by unity (except when p indicates a Pawn, Knight and King). Then the machine finds the number s of the square from which the last move had been made and the previous direction counter j_d. By means of these numbers, the machine forms the sum $t=s+d \times j_s$; this is the number of the square reached by the new move (which goes one step further than the previous one in the same direction).

Having found t, the machine looks up the b-table to examine the contents of this square. It may turn out that this square is one of the border squares, or that it is occupied by another white piece, in which case the move has been found to be impossible. We shall deal with this case later and assume for the moment that the new square is either empty or occupied by a black piece *i.e.* that the machine has found a possible move*. Under the control of the symbols for 'black' or 'empty' which the machine has found in line $b+t$, it now enters a routine to test whether the move is legal *i.e.* to ensure that the move has not put the white King in check. This is a complicated process the description of which must also be deferred for the moment. If the move turns out to be illegal the machine returns to the process of counting up and trying out the next move. If it is legal then the new position (possibly also modified by the taking of the black piece on square t) is recorded by suitable entries in the a-table and the b-table, and both these tables, together with the new values of the counters, are transferred to the magnetic store, Later, the new position will be used as a starting point for Black's first-turn moves, while the new set of counters will indicate the next move to be made when it

*In cases of Pawn moves the criteria for possibility are suitably modified to conform with the rules of the game.

is White's first turn.

If desired, the machine can be made to print every white first-turn move on its output type-writer. This is useful to ensure, at regular intervals, that the machine is still in working order.

Let us now return to the case where a tentative move made by the machine is found to be impossible, as indicated by the symbols ('border square' or 'white') found in line $b+t$. The machine will then increase the direction counter by a number depending on the piece *e.g.* 1 for King and Queen, 2 for Bishops and Rooks. Again, there are two possibilities. If the new direction counter does not exceed its maximum value 7, the machine will try a single-step move in the new direction and then proceed as before. If j_d does not exceed this number, indicating that all possible directions for this piece have been tried out, the machine is made to enter another sequence which increases the piece counter j_p by 1. Once more, we have two different cases. If the new j_p exceeds 15, *i.e.* if all the 16 pieces have been dealt with, then the position is exhausted; for the white first-turn moves, this means that the problem has no solution at all and the machine will be made to stop or to print out a remark to this effect. If, on the other hand, the new j_p does not exceed 15 then the machine examines line $a+j_p$ in the a-table. Here the process divides again. If the contents of this line indicate that the piece with the counter j_p is not on the board at all, the machine will increase j_p again and go on as before. If the piece is on the board, then line $a+j_p$ will contain the number s of the square containing this piece. The machine will then record j_p and s in their proper storage lines, and will look up line $b+s$ in the b-table where it finds the piece number p. This is also recorded and the machine goes back to the process described initially, *i.e.* to a sequence constructing a new move appropriate to the piece p, but in this case a single-step move in the first of the possible directions.

The same processes occur in the other three turns of the game, and it is only necessary to add that the first move in any turn is preceded by resetting the counters to their initial values.

The routine for testing legality uses techniques similar to that for finding the next possible move. Assuming we are in a white turn, the machine starts by examining the contents of line $a+15$ where it finds the square on which the white King stands. From this square as a starting point, the machine then explores the board by a series of imaginary moves, leading to various squares t. By looking up line $b+t$ in the b-table, the machine then finds out whether t is outside the active board; if not, whether it is empty; and if not, whether the piece standing on it is white or black. Except for the last case, the machine will then immediately construct the next imaginary move. In case of a black piece, the machine examines whether the rules governing this piece are in such a relation to the move just made that the King is under attack. Thus, a black Rook can endanger the white King only if the imaginary move just made had an even j_d: for the black King, j_s must have been 1, and similarly for the other pieces. (Knights are treated in a slightly different way; the machine makes imaginary Knight's moves and tests whether or not the value of p, found in line $b+t$, equals 3, the piece number allocated to Knights).

The move is illegal if a piece attacking the King can be found; it is legal if the process exhausts the position without finding such a piece. As mentioned before, the need for scanning the greater part of the board whenever a move is made makes this process one of the most time-consuming operations in the programme.

We have seen that the programme for the solution of mate-in-two chess problems must contain a routine for the construction of the next possible move, a routine to check this move for legality, and various sequences for recording the moves made and the positions obtained. All these pieces must now be linked together by a master routine reflecting the structure of the problem as a whole, and ensuring that the various turns are entered in the correct sequence.

This structure is illustrated by the diagram in *Figure 4.* Each of the 4 blocks marked W1, B1, W2, B2 represents one of the turns. Each block is provided with two 'entries' 1, 2 and two 'exits' 3, 4. Entry 1 corresponds to the case of the first move in a turn, with all the counters set to their initial values. Entry 2 is the general case of a move following a previous move of the same turn. Exit 3 indicates that a legal move has been found; exit 4, that the position supplied to the turn has been

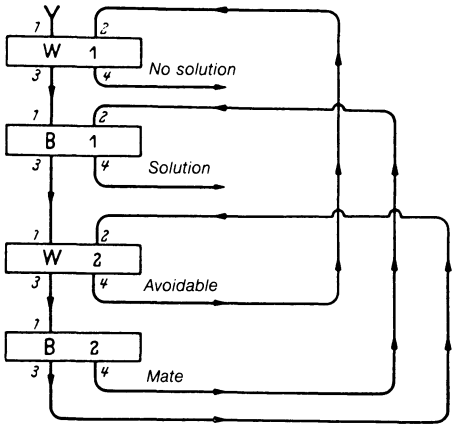

Figure 4

exhausted before such a move has been found.

Let us look at the diagram in detail. Right at the beginning, turn W1 is entered at 1. Normally, it will find a legal move, and its exit 3 will lead to entry 1 of B1, the first move in Black's first turn. If B1 finds a legal move, its exit 3 will lead to entry 1 of W2, and similarly, a legal move of W2 will lead from its exit 3 to entry 1 of B2.

If now B2 finds a legal move (exit 3) then White must try another second-turn move, *i.e.* exit 3 of B2 must go to entry 2 of W2, but if B2 finds no legal move, Black is mate (or stale-mate) and must try another first-turn move. This means a connection from exit 4 of B2 to entry 2 of B1.

If, after a first-turn move by Black, White tries out all its second-turn moves without achieving mate, then White must try another first-turn move: this gives the connection from W2, exit 4, to W1, entry 2. On the other hand, if after a W1 move Black exhausts the position without finding a move avoiding mate, then this move by White was the key move; for this reason exit 4 of B1 is marked 'solution'. No solution exists if W1 tries all legal moves without finding one leading to an unavoidable mate; this is exit 4 of W1.

Once the structure of the problem has been translated into numerical form, it is not difficult to arrange a scheme causing the machine to produce a set of suitable numbers at an exit from a turn, and then to interpret these numbers to find the correct entry turn for its next operation. This scheme is carried out by the master routine. The by now familiar technique of looking up and modifying tables is again employed, and it will hardly be necessary to go into further details. Moreover, I trust that this brief outline will have been sufficient to give the reader an indication of the methods that might be used to treat structural or logical problems occurring in other fields by means of electronic computers.

6 Writing a Chess Program

A whole book could be devoted to this one subject. In fact there are already at least two such books in print (Spracklen et al. 1978, and Levy 1983). In the present chapter there is a need for selectivity, and so I have restricted myself to three papers.

Norman Whaland's tutorial (section 6.1) provides a useful background for anyone wishing to write their own program. In particular the flow chart of the alpha-beta algorithm is extremely useful.

David Wilkins describes how chess knowledge can be used in the search process (section 6.2). His program PARADISE uses chess knowledge to discover plans during the analysis process and to guide the search in order to confirm that a particular plan is best. Computer chess enthusiasts may well have read complaints from strong human chess players that programs lack ability to plan, and it is well known in chess that "even a bad plan is better than having no plan at all!" I would strongly recommend that anyone wishing to write his own program should make a special study of this paper, in conjunction with Harris' work on heuristic search. There is a wealth of good ideas to be explored, more interesting and possibly more rewarding than those employed in writing a "brute force" program.

The paper by John Birmingham and Peter Kent (section 6.3) describes how a program can recognise a mating situation before the mate has occurred. This can be of great use in the tactical search, since a mate threat can be examined further to determine whether or not it is part of a successful variation. I would argue in favour of the inclusion of such a routine in any chess program – after all, the object of the game is to force mate.