Chess Circuits

Ross Dempsey Sydney Timmerman Karl Osterbauer

March 9, 2018

Abstract

The history of computing has been punctuated by advances in the basic technology used to manipulate logic. Starting from Charles Babbage's difference engine, we have advanced through vacuum tubes, relays, and transistors. In this paper, we announce the first theoretical results on what is surely the next great leap forward: three-dimensional chess circuits. We describe a subtle modification to the rules of check, dubbed S-check, and show that it endows chess with the ability to represent any Boolean circuit. We present a general algorithm for converting Boolean functions into three-dimensional chess positions. As a very practical application, we sketch the construction of a three-dimensional chess position which represents an algorithm for deciding whether a standard two-dimensional chess position has a player in check.

1 Introduction

Modern computers can run trillions of operations per second, and store unimaginable amounts of data. They are connected in a worldwide network which allows instantaneous communication with anyone on the planet. Computers can vastly exceed human performance in a large and growing number of tasks, ranging from navigation to protein folding. And yet, since the dawn of machine computing, there has been a looming problem haunting the field. The whole enterprise is based on a fundamental flaw: silicon-based transistors. Silicon is an ugly semimetal.

Compare silicon with the smooth, dark luster of a mahogany chess board. Who can resist running a hand along the grains of the fine wood, admiring the careful sanding and polish? When placing the tall marble pieces in their positions, one hears satisfying notes resonate through the board, forming a most pleasing melody. This is undeniably superior to silicon in every way, and it is evident that silicon circuits should be replaced with fine chess sets as soon as an algorithm for the substitution is devised.

In this paper, we present such an algorithm. Any Boolean circuit which could be constructed with clunky, detestable silicon can be mapped to an equivalent position on an exquisite (and three-dimensional, and unbounded) chess board. Through a modified definition of check, known as S-check, the Boolean function computed by the vile silicon mess is instead evaluated in a civlized manner: by determining whether a bishop is capable of delivering S-check.

2 Three-Dimensional S-Chess

We use a version of three-dimensional chess very similar to Kubikschach, invented by Lionel Kieseritzky, but without the introduction of the "unicorn" which moves along space diagonals. Rooks move in directions (1,0,0), (0,1,0), and (0,0,1). Bishops move in directions (1,1,0), (1,0,1), and (0,1,1). Kings and queens move

in all six of these directions. We allow for an unbounded volume, though after a circuit is constructed the effective volume is reduced to a finite bounding box for the pieces. We also allow an unlimited number of every type of piece.

The important distinction we make is in the rules of check. Consider the chess position in Figure 1, with white to move. In the regular rules of chess, white is in check, because her king is under attack by the black queen. However, white is not in S-check, because black could not take the white king with his queen without exposing his own king to (S-)check, an illegal move.

Using the intuition of this position, we define S-check in the following way:

Definition 1. *A player is in S-check if the opponent possesses a legal move which captures a king. A move is illegal if it leaves the mover in S-check.*

This is a stronger condition than standard check. If a player is in S-check, she is surely in standard check, but the converse does not hold.

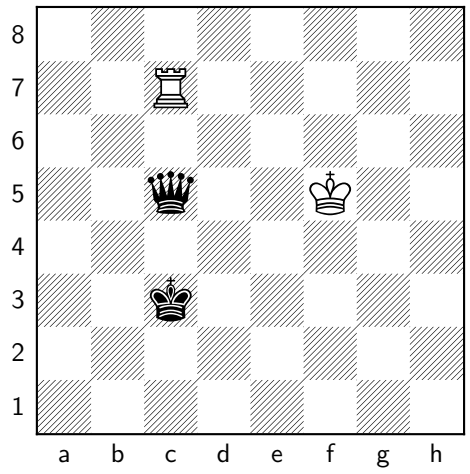

Figure 1: White is in check, but not in S-check.

3 Bishop NOR Gates

Every piece in chess is either pinned to a king or not. We use this property to store bits on a chess board. A piece which is pinned is a 0, and a piece which is free to move is a 1. Consider the position in Figure 2 in this context. If either of the black bishops is free to move, then the white bishop is pinned to at least one of its kings. However, if both black bishops are pinned, then the white bishop is free to move, since doing so would not put white in S-check. The white bishop thus represents the NOR of the two black bishops.

Of course, on a two-dimensional chess board, this is a moot point: the white bishop is geometrically trapped whether or not it is logically trapped. It is for this reason that we introduce a third dimension. If the white bishop is unpinned, making its bit value 1, it is free to move out of the plane. Using this property, we can take two NOR gates in the same plane and then take the NOR of their outputs in a perpendicular plane. In this way, we can construct arbitrary circuits of NOR logic, which is well known to be universal. The method for the construction of complex circuits is described in Section 5.

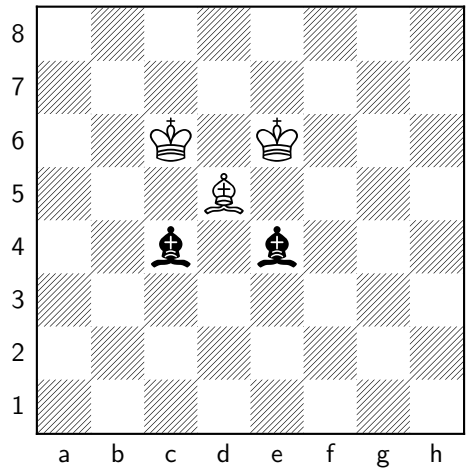

Figure 2: The white bishop implements a NOR of the black bishops.

4 Rook Memory

A circuit is useless without a way to input values. Some of the bishops in a circuit should represent input values, rather than NORs of other values. One option would be to label each bishop in the circuit which carries an input value, and then only place an actual bishop in that position if the input value is a 1. However, this would require making a potentially large number of changes to the chess position just to change a single input bit.

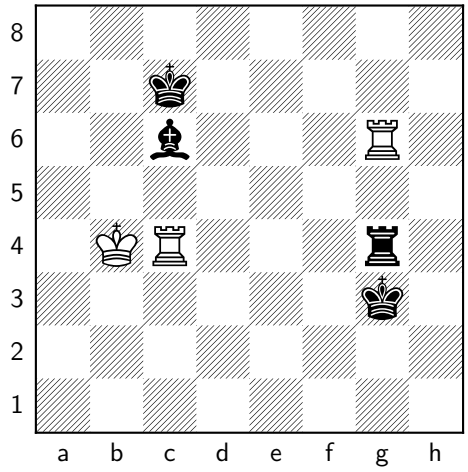

Figure 3: Rooks carry a value in memory to a bishop in the circuit.

Instead, we store memory in rooks. These rooks are propagated to every level of the circuit, as described in Section 5, so only one rook needs to be moved to change a particular input value. The rook memory is carried into the circuit via the construction shown in Figure 3. Let A be some Boolean variable we wish to retrieve from memory. The white rook on g6 is assumed to carry the value $\neg A$. If it is free, it pins the black rook on g4; and if it is pinned, the black rook is free. Thus, the black rook on g4 carries A. Likewise, the white rook on c4 carries $\neg A$, and the black bishop on c6 carries A, as desired.

5 Circuit Building

Any Boolean function can be converted into NOR logic. For example, the typical AND and OR operations can be represented by

$$\begin{aligned} A \text{ AND } B &= (A \text{ NOR } 0) \text{ NOR } (B \text{ NOR } 0), \\ A \text{ OR } B &= (A \text{ NOR } B) \text{ NOR } 0. \end{aligned} \tag{1}$$

Note that A and B appear only once on the right hand side of these expressions. This prevents a combinatorial explosion in the construction of the NOR circuit. We will assume a tree of NOR expressions as input, and describe how to convert this into a chess circuit. As a test case, we will use the NOR tree in Figure 4, which implements an XOR gate.

The "base case" is trivial. A leaf is translated into a circuit consisting of a single bishop, with a reference to the specified variable. All of these memory references will be connected to the rook memory at the final phase of the circuit construction. We also need to be able to convert NOR nodes in the tree into circuits.

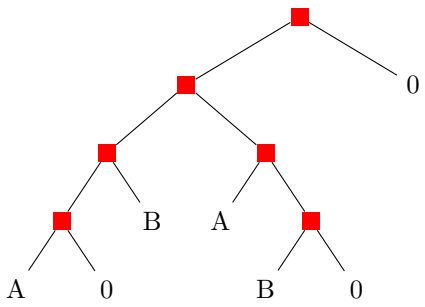

Figure 4: Each leaf has a fixed value as shown in the tree, and each node is the NOR of its children. The root node is A XOR B.

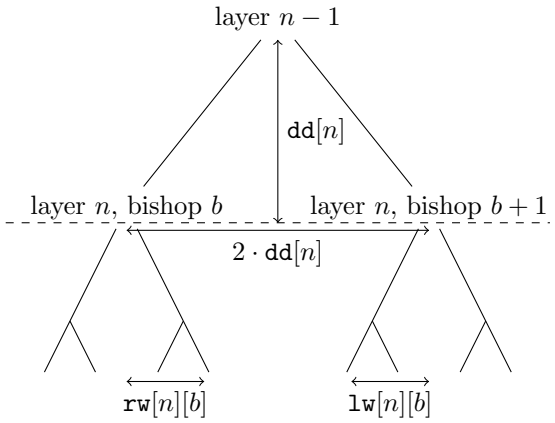

Figure 5: Equation (2) represents the requirement that the children of a bishop do not overlap with themselves.

Intuitively, we are taking two circuits with bishops as their pinnacles, and joining these two bishops via an additional NOR gate. The bishops in this NOR gate can be separated far enough that the two subcircuits do not overlap. However, there are several possible complications in this process.

- **Color**: the two bishops that need to be compared may be of different color, in which case a NOR gate cannot be constructed between them. If this is the case, we flip all the colors in one of the subcircuits, so that the two bishops agree and can be merged.
- **Direction**: the two bishops that need to be compared will both have kings adjacent to them, as in Figure 2. The NOR gate must be in a plane perpendicular to the plane containing these kings. However, the two bishops may not *a priori* share such a plane. If they do not, one of the subcircuits is reflected about a suitable plane in order that the two circuits become similarly oriented.
- **Homogeneity**: the two subcircuits may have different shapes. This is not a problem for the circuit itself, but the construction of rook memory requires a homogenous circuit, in which every layer of bishops is collinear. This can be achieved in a merger of circuits in two phases. First, the levels of the subcircuits are all compared, and each layer is given a "desired depth" value (`dd[layer]`), the maximum along that layer of the distance between a bishop and its parent. Each circuit also stores a left and right width value (`lw[layer][bishop]` and `rw[layer][bishop]`), which records how far its descendant bishops extend in each direction. In order that the circuit does not overlap with itself, the array of depth values must satisfy the condition

$$2 \cdot \texttt{dd[n]} \geq \max_b(\texttt{rw}[n][b] + \texttt{lw}[n][b+1] + 1). \tag{2}$$

  This inequality is represnted in Figure 5. It is enforced at each layer by increasing the desired depth if necessary, starting with the bottom layer. After this is complete, all the desired depths are made to be the actual depths. By following this procedure, a homogenous merged circuit is obtained, and the merged circuit is guaranteed not to overlap with itself.

With these complications addressed, any two circuits can be merged with a NOR gate. Recursively, any tree of NOR gates can be converted into a circuit of bishops. Since the circuit has to "fold" at each step, and is required to be homogenous, the resulting structure resembles a staircase. Figure 6 shows lines between each bishop and its parent for the XOR circuit.

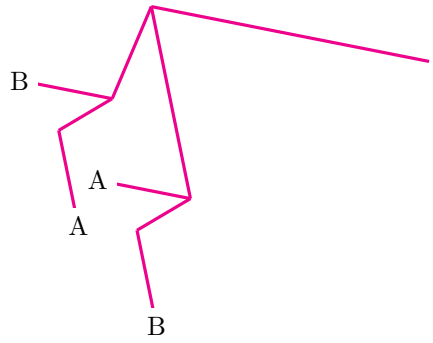

Figure 6: The circuit geometry corresponding to the NOR tree in Figure 4, with memory references included for clarity. Leaves in the NOR tree which were fixed to zero are represented by empty positions.

After the circuit is constructed in this way, we add the rook memory. Let N be the number of inputs to the circuit. We reserve $2N$ positions adjacent to the bottom layer of bishops. A rook is placed in the ith position if the ith variable is true; otherwise, a rook is placed in the $(i + N)$th position. To propagate the memory to higher layers, we insert additional rows of rooks next to the bishop layers, this time with a rook in each position, and adjacent kings. This pattern carries the variable A at one layer to $\neg A$ in the corresponding rook of the next layer. Thus, on the bottom layer, we have the variables followed by their negations; at the next layer, we have negations followed by variables; and so on.

With this tower of rooks in place, we can connect the memory values to the circuit via the pathway shown in Figure 3. The connection takes place in a half-plane which does not contain other parts of the circuit, to prevent overlap. This is the final step in the circuit construction. Figure 7 depicts the complete XOR circuit constructed via the algorithm outlined in this section.

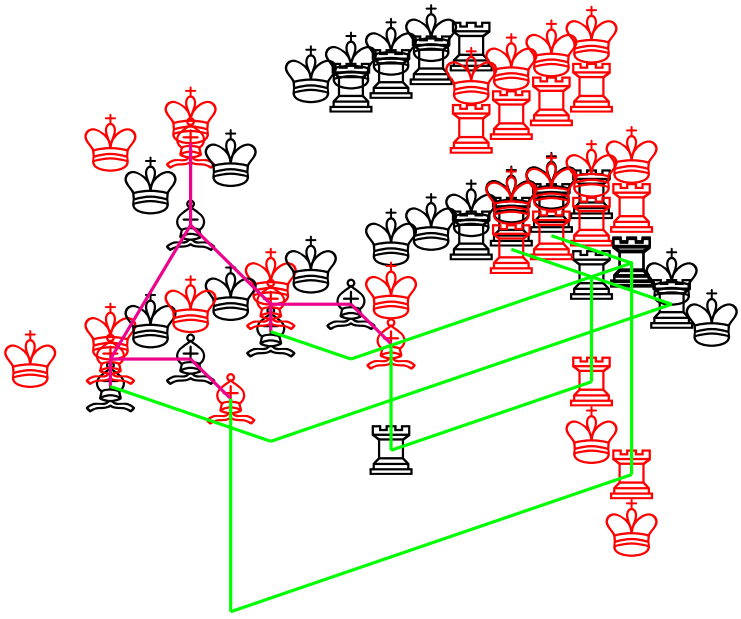

Figure 7: The complete chess circuit implementing the logic in Figure 4. Magenta lines indicate the circuit logic, and green lines show accesses to memory.

6 Application

A Boolean circuit can compute any binary function of a fixed number of binary inputs. Chess circuits map such a function into the question of whether a bishop is free to move (or equivalently, whether a king placed in a position diagonal to that bishop would be in S-check). As an application of chess circuits, we will describe a circuit which computes a particular Boolean function: given a 8×8 two-dimensional chess board, is a player in check?

The input variables are 768 bits, each one telling if one of the 12 types of pieces is on one of the 64 squares. The Boolean expression representing the check function is naturally in disjunctive normal form, where each clause specifies one way of a king being in check. For example, one clause would take the form

$$(kc6 \wedge 0c5 \wedge 0c4 \wedge Rc3).$$

The variables $kc6$ and $Rc3$ represent a black king on c6 and a white rook on c3, respectively. The terms $0c5$ and $0c4$ are abbreviations for squares c5 and c4 being unoccupied; these in fact represent conjunctions of twelve negations, such as $0c5 = \neg(pc5 \vee Pc5 \vee nc5 \vee Nc5 \vee \cdots)$. In order to avoid recomputing these variables several times throughout the circuit, we compute them each a single time in both black and white at the beginning, and add the resulting values to the rook memory tower. Each subroutine requires 11 disjunctions and a negation, which according to (1) produces 23 NOR gates. There are thus a total of $23 \cdot 64 \cdot 2 = 2944$ NOR gates in the subroutines.

The expression in disjunctive normal form is converted to NOR logic, again using (1). A king can be in check in 4690 ways, leading to $2 \cdot (4690 - 1) = 9378$ NOR gates. A careful counting of the conjunctions in each clause leads to a total of 9120, for $3 \cdot 9120 = 27360$ NOR gates.

The grand total for the circuit is $2944 + 9378 + 27360 = 39682$ NOR gates. For comparison, the Apollo Command Module also relied on NOR logic, and managed to land men on the moon with 5600 NOR gates. However, they used silicon, which is ugly. Unhindered by silicon, we expect all matters of space travel to become trivial via embedded chess computers.

7 Conclusion

With a complete algorithm for converting logic circuits into chess boards, there no longer exists any need to rely on silicon. Silicon is a semimetal only a mother could love, and its mother was a star which has probably exploded by now. Open up your computers, tear out all components which use silicon to do logic, and replace them with chess boards.

Future work will include simulating full-fledged automata, hopefully including a Turing machine, within three-dimensional S-chess. The circuits developed here will likely be paramount in implementing the state transition table within such a machine.