Turing-Complete Chess Computation

Ross Dempsey Sydney Timmerman Karl Osterbauer Kat Xiang

April 1, 2019

Abstract

Just one year ago, the course of history was dramatically altered by the introduction of the most aesthetically pleasing mode of computation ever conceived, the three-dimensional chess circuit. Since that date, the chess computing community has been grappling with the difficult question of why chess circuits have not yet been universally adopted. In this paper, we present the results of a comprehensive survey of reasons why chess circuits are not being used more widely. After excluding an outlier response, 'WTF,' we find that the primary concern with chess circuits is the difficulty of incorporating them into a full Turing-complete model of computation. We respond to this concern with the design of a Turing machine within a three-dimensional game of S-chess.

1 Introduction

Before light, there was the darkness. Before music, there was the silence. Before sunshine, there was the storm. And before the chess circuit, there was the silicon.

It belongs to no family, no creed, no nation. Neither metal nor insulator, it is an unspeakable mutant, a rotten semimetal. Spending its days in back alleys, getting doped up, its destiny was to die alone, master of none. And yet we enslaved ourselves to it, and billions of humankind bent their backs and twisted their wrists yearning for its false allure. Its unholy seductive power brought the whole Earth under its dominion, and each of us prostrated ourselves, worshipping this false idol. It is silicon, the computing technology which must not be named.

Only a year ago, the authors unshackled humanity from its bondage and preached the gospel of the chess circuit [1]. Like a chorus of angels, ecstatic children filled the streets with their cries of relief; for the dark

Figure 1: Devout churchgoers kneeling in worship of the fine mahogany which sits immediately in front of them.

days were over, and the light was upon us. Mighty silicon gave way to the wise hand of mahogany. All across the world, people come every Sunday to be held in seats of wood, made to represent the true fine mahogany. In moments of need, they kneel before the wood in front of them, in worship and admiration of mahogany (Figure 10).

Despite the resounding show of support for fine mahogany, silicon retains a great deal of its power over the world. In this paper, we seek to understand why there has been resistance to the adoption of a clearly superior technology. To this end, we conduct a survey of reasons for the continued use of silicon circuits over chess circuits. After excluding a prominent outlier response, 'WTF,' we find that the predominant reason is the difficulty of constructing a full computer out of chess circuits. The remainder of the paper addresses this concern by constructing a chess Turing machine.

2 Survey Results

We conducted a survey of 100,000 respondents to understand their feelings about the adoption of chess computers. While normally it would be difficult to tally the responses of such a large number of participants, we used a chess computer, and so the results were available in a blazingly fast 3.5 months. (Three additional weeks were spent waiting for an order of 80,000,000 additional white rooks, since we needed to store 10 MB of data).

The survey asked a single question: "what would prevent you from replacing all of your silicon-based devices with chess-based equivalents?" The results are summarized in Table 1. Although the survey was open-ended, we only received two distinct responses. One was the three-letter string "WTF," and the other was the sentence "I am not sure how to incorporate the static Boolean circuit capability outlined in [1] into a dynamic Turing-complete system which can fully service my computing needs." This sentence was repeated verbatim four times, and there are four authors of this paper. However, since the survey was anonymous, we cannot investigate this coincidence further.

Response	Count
WTF	99,996
I am not sure how to incorporate the static Boolean circuit capability outlined in [1] into a dynamic Turing-complete system which can fully service my computing needs.	4

Table 1: Our free-form text response survey received only two distinct responses, which are summarized here.

We have reviewed numerous possible explanations for the occurrence of the string "WTF." One possibility is that these survey participants are intending to say "Wow, That's Fun!" in response to the idea of replacing silicon circuits with chess circuits. Regardless of the explanation, we believe it is safe to treat these 99.996% of responses as outliers, and exclude them from our analysis.

After making these reasonable adjustments to our data set, we find that an overwhelming 100% of responses indicate a lack of confidence in building a Turing machine out of chess circuits. In this paper, we respond to these concerns with the design of a Turing machine which runs on a three-dimensional game of S-chess.

3 Three-Dimensional S-Chess

In case any of our readers are absolute fools who have been living under a rock for the past year, we review the rules of three-dimensional S-chess. We use a unicorn-free version of Kubikschach, invented by Lionel Kieseritzky. Rooks move in directions $(\pm 1, 0, 0)$, $(0, \pm 1, 0)$, and $(0, 0, \pm 1)$. Bishops move in directions $(\pm 1, \pm 1, 0)$, $(\pm 1, 0, \pm 1)$, and $(0, \pm 1, \pm 1)$. Kings and queens move in all of these directions. We allow for an unbounded volume, and 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 2, 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.

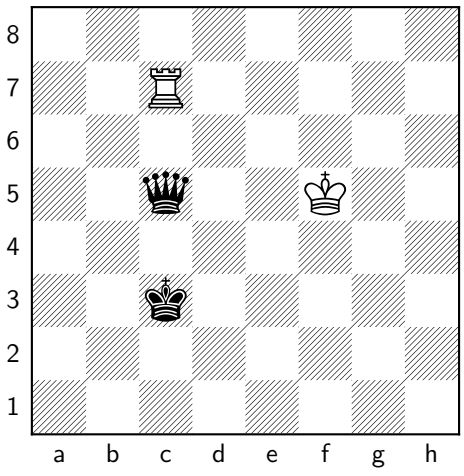


Figure 2: White is in check, but not in S-check.

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.

Since there are multiple kings of each color, it is possible for multiple kings to be in S-check at once. This would normally result in what we call a "lame checkmate," where a player wins the game by forking her opponent's kings. We prevent lame checkmates by allowing a player to make N moves per turn, where N is the number of kings which are in S-check.

4 Chess Circuits

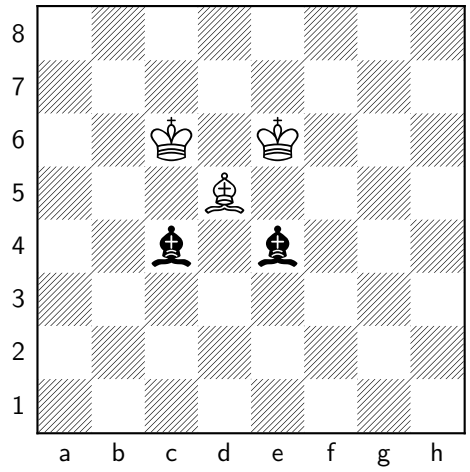
In [1], we present an algorithm for building chess circuits corresponding to any Boolean function. Here we review the basic constructs.

Bits in the chess circuit are represented by the binary property of whether a piece is pinned or not. A piece which is not pinned, and therefore free to move, is assigned a 1; a piece which is pinned, and therefore fixed in place, is assigned a 0.

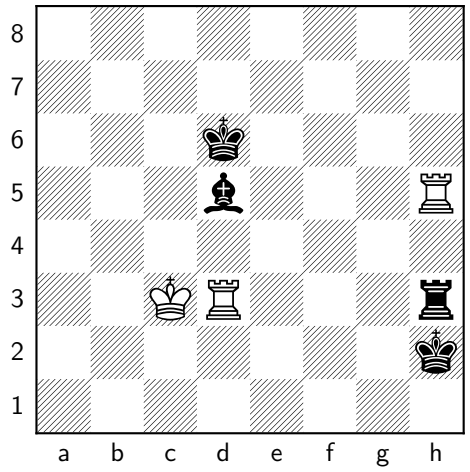
To build circuits, we use bishop NOR gates. An example of a bishop NOR gate is shown in Figure 3a. The two black bishops are part of a larger configuration, so they may be 0s or 1s. If they are both 0s, i.e. both pinned, then the white bishop is unpinned and takes the value 1. If either black bishop is unpinned and has value 1, then it pins the white bishop, which has value 0. Therefore, the white bishop is the NOR of the two black bishops.

Since NOR logic is universal, we can put these gates together into an arbitrary Boolean circuit. Each layer of gates has to be rotated 90° with respect to the last, so a staircase pattern results. The details of the geometry are outlined in [1]. The result of the circuit is stored in a bishop which sits at the apex.

In addition to the circuit itself, we need to insert the values of the variables. A single variable may appear at



(a) The white bishop implements a NOR of the black bishops.



(b) Rooks carry a value from memory to a bishop in the circuit.

Figure 3

many points in the circuit, so we have to be able to wire the same truth value into several bishops, possibly at many levels of the circuit. To do this, we construct a tower of rooks next to the circuit, which follow the same staircase pattern. To move a value from the rook tower into the circuit, we use a rook wire like the one shown in Figure 3b.

5 Chess DFA

The aim of this paper is to construct a Turing machine out of the rules of three-dimensional S-chess. We will do this in two phases. First, in this section, we describe the construction of a deterministic finite automaton (DFA) in S-chess. This illustrates many of the important features of the S-chess Turing machine, but without the complications of a writable memory tape. Then, in Section **??**, we extend the DFA construction to make it Turing-complete.

A DFA consists of N states, which it moves between depending on the bits of an input string. We will take $N = 2^n$, so that states can be labeled by bit strings of length n. One state is designated as the accept state, and the string is accepted if and only if the machine finishes in the accept state.

Typically, the work of a DFA is represented as a walk on a graph of states. We will instead use repeated applications of a state transition function $\Phi : \{0,1\}^{n+1} \to \{0,1\}^n$. This function takes a state, represented as a bit string of length n, and an additional bit, and returns another state. Let the initial state be i, and let the input string be $s_1 s_2 \cdots s_m$. Then the final state of the DFA is given by

$$\Phi(\Phi(\cdots \Phi(\Phi(i, s_1)s_2) \cdots, s_{m-1}), s_m).$$

The key point of our construction is that $\Phi : \{0,1\}^{n+1} \to \{0,1\}^n$ can be represented by n Boolean circuits, each of which take $n+1$ inputs. Thus, we can compute the state transition function using an array of n chess circuits.

The input to these chess circuits will be n bits specifying the current state, and one bit from the input string.

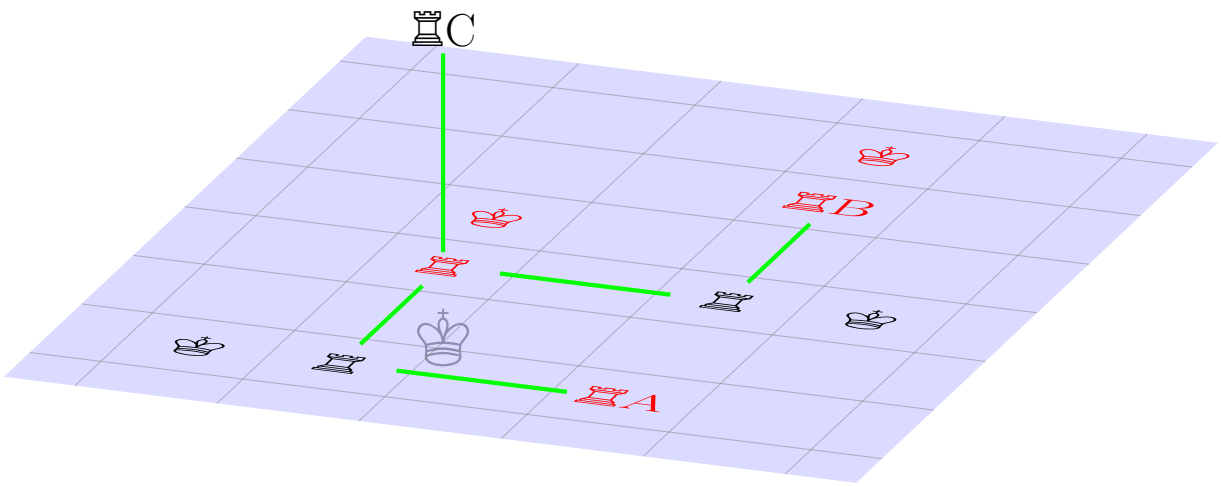


Figure 4: A chess transistor, in which a control rook regulates the flow of data through a wire.

Providing these two inputs to the circuits will constitute the primary challenge in constructing the chess DFA. We will first describe the mechanism for storing the current state, and then describe the input reader.

5.1 Internal Memory

The output of the state transition circuits has to be fed in as input at the next time step. We cannot wire the output to the input directly, since there would be no time to pause for the next bit of input (and also there would be many issues related to placing oneself in check). Instead, the bits need to be copied into some form of storage, and then piped into the circuits at a later stage.

This reveals a fundamental difficulty with constructing a dynamic chess computer as compared with constructing a static chess circuit. In a chess circuit, if we want to move a bit, we simply build a rook wire; in a chess computer, we have to take great care to allow for a steady progression of time, without copying values too quickly. To fix this problem, we will need to be able to have wires which are activated or deactivated according to the value of some other bit. In effect, we need to build a chess transistor.

As a toy example, let there be a rook storing a value A, and a control rook storing a value C. If $C = 0$, we want the rook B to take the value A, and if $C = 1$, we want $B = 0$. Of course, this could be accomplished using our standard construction of chess circuits, but there is a more efficient option. Figure 4 shows an implementation of the desired behavior.

Using the chess transistor, we can build a circuit for gated data flow. We will need a slightly different convention for storing bits – since pins can be deactivated by transistors, they do not persist. Instead, we will use the position of a piece to store internal memory. It is straightforward to translate one data type into the other, by (i) pinning a piece depending on the position of another piece or (ii) forcing a piece to move to one of two positions to defend against an S-check created by the loss of a pin. Note that type (i) copies are instantaneous, while type (ii) copies require the passage of a turn.

The gated data flow circuit for a single bit is shown in Figure 5, using rooks everywhere for simplicity. The initial data, to the left of the figure, is represented by the placement of the rook in one of the two indicated positions at location (a). This pins exactly one of the rooks directly below it, at location (b). These rooks are connected by gated wires to another set of rooks, at location (c). When the control bit C for the gated wires is on, the leftmost two rooks at location (c) are 0, and so the leftmost two kings at position (d) are

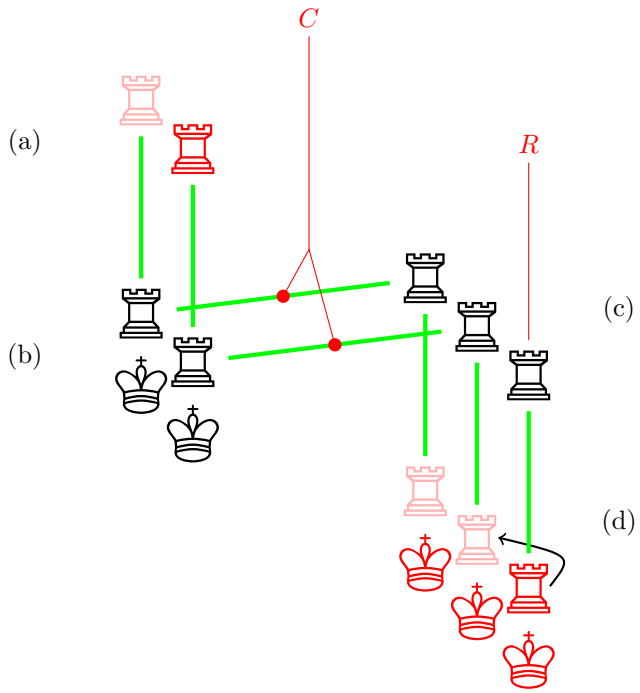


Figure 5: A gated data flow circuit, which uses transistors to move data from (a) to (d) over the course of a turn.

not in S-check. However, when $C = 0$, data flows to position (c) and so exactly one of those rooks carries a 1, which then places one of the leftmost two kings at position (d) in S-check. The rook at position (d) has to shift to block this S-check.

Thus, when the control bit C is activated by black, white is forced to copy a bit from position (a) to position (d) in the next turn. In order to ensure that white remains in perpetual check, it is important to reset the circuit using the control bit R before the next copy, forcing the rook at (d) to move back to the rightmost position.

5.2 Read-Only Memory

In the DFA, the input string is read-only, and is used in order one bit at a time. This is the distinction between the DFA and the Turing machine. The mechanism described in this section is the first stage in the construction of a more complicated memory system for the Turing machine.

We store the input string as a line of rooks, where the presence of a rook corresponds to a 1 and the absence of a rook corresponds to 0. Adjacent to the line of rooks, we place another line of rooks of the opposite color, with a single gap. Whenever a rook in this second line line moves, the gap moves to the next position in the input string. Figure 6 shows this construction.

This allows us to isolate one bit of the input at a time, but the bit of interest is moving. We need to build an apparatus around the input string which channels this bit into a fixed position. In Figure 6, when the gap is at position i, all of the white bishops have value 1 except for the ith, which has value $\neg s_i$. If we build a circuit to compute the conjunction of all these bishops, then the value at the apex will be $\neg s_i$. This value can then be piped into our n state transition circuits via rook wires.

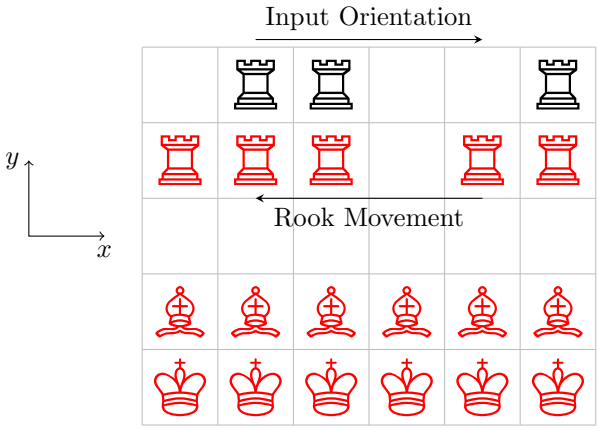


Figure 6: As the rooks move, the gap moves in the opposite direction, exposing one bit of the input string at a time.

We still need a way to force the white rooks in Figure 6 to move in the specified direction. For this, we use a line of black rooks above the white rooks, as shown in Figure 7. The rooks are wired in a repeating threefold pattern, which we have labled with variables X_0, X_1, and X_2. During other stages the machine operation, we set all X_i to 0. But when it comes time to move the memory, all but one of the X_i are set to 1. The variable which is set to zero is controlled by a counter which we update at every time step using the methods of the previous subsection. This way, we can track the position of the memory gap and act accordingly. The black rooks are set so that white is in check, and can only defend by moving a rook such that the memory gap moves in the desired direction.

5.3 Construction

We now have all the basic circuits and mechanisms we need to build the chess DFA. Our only job is to put them together correctly.

Recall that the main computation of the DFA with 2^n states is carried out by n chess circuits, each with $n+1$ inputs. We will denote these circuits by $A_1, \ldots, A_n$.

We arrange these circuits side-by-side, and simultaneously homogenize them all. While this is not strictly necessary, it allows us to use a single rook memory tower to source all of the circuits. Some circuits may be deeper than others, so the memory tower starts at the bottom of the deepest circuit. See [1] for a detailed discussion of chess circuit homogenization.

In addition to the circuits $A_1, \ldots, A_n$, we add three small circuits which keep track of the time step modulo 3, for use in the memory movement mechanism. These circuits are labeled B_0, B_1, and B_2.

The chess DFA has to make the following steps in each time step:

1. Evaluate all circuits A_i and B_i on the input $(\sigma, s_i, b_0, b_1, b_2)$, where σ is the current state, s_i is the current bit of input, and b_i are extra variables representing a counter modulo 3.
2. Store all the results in internal memory.
3. Move to the next bit of memory.

To accomplish these steps in sequence, we make ample use of chess transistors, controlled by a central clock

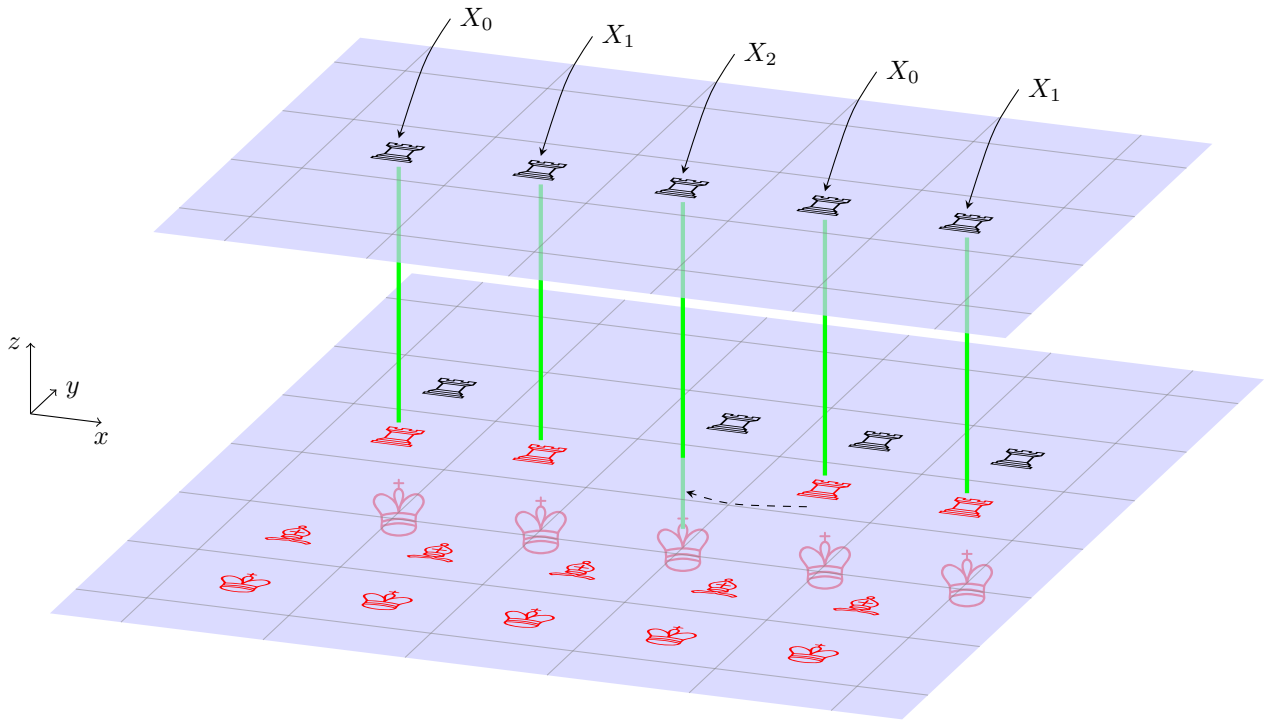


Figure 7: During the memory movement phase, all but one of the variables X_i are set to 1, so that white's only choice for defending check is to move a rook. In this figure, X_0 would be set to 0 and the others set to 1, so that white would have to move its rook to the left and defend, thereby moving the memory position to the right.

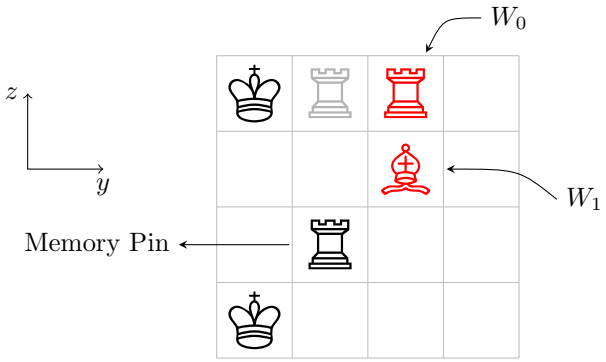


Figure 8: By activating exactly one of W_0 and W_1, we can write either a 0 or 1 to memory respectively.

circuit which consists of a queen moving in a polygonal path. On a detailed level, the following steps need to be taken in each time step:

1. A transistor is opened so black bishops at the apexes of circuits can put white kings in check, forcing white to move rooks to defend.
2. A gated wire is opened, copying these white rooks to another set of white rooks.
3. The first white rooks are reset.
4. The black rooks in the memory movement mechanism are activated, moving the memory gap.

Each of these steps consists of black freely moving the queen in the clock circuit, followed by white responding by defending against all of its checks. Thus, each step consists of a single turn (where both players move once in a turn), so a time step of the DFA corresponds to four turns of the chess game.

Constructing a chess DFA requires wiring the clock circuit to transistors such that steps 1-4 are carried out in sequence each time step. Rather than sketching the DFA layout, we will first describe the modifications necessary to build a full Turing machine, and then sketch this.

6 Chess Turing Machine

A Turing machine differs from a DFA in how it interacts with memory. In a DFA, there is a fixed input string which is read bit by bit, and the final state of the machine is the output. In a Turing machine, we can move back and forth through the memory, and also write to memory.

Upgrading our DFA to a Turing machine thus requires two modifications. First, the machine should be able to move in either direction along the memory tape. This is surprisingly simple given the construction described in Section 5.2. We simply add an additional circuit to the core of the machine, M. When M outputs 1, follow the same procedure as in Section 5.2, moving the memory gap to the right. When M outputs 0, we set X_0, X_1, and X_2 such that the memory gap moves to the left instead.

Additionally, the machine should be able to write to memory. This is also surprisingly easy. For the DFA, memory was stored by a sequence of rooks and gaps; for the Turing machine, we replace the gaps with rooks that have been forced to move out of place to defend against a check. At each time step, we can alter the position of this rook by placing a king in check and forcing it to defend.

There are two small caveats when writing to memory. The first is the geometry; we have to be careful to move the rook to a position in which it will be "out of sight" to the memory system, while also not blocking the path by which the in-place rook pins a bishop. Figure 8 shows a suitable geometry, using a white rook and a white bishop.

Figure 8 only shows the yz plane; it is also crucial that this mechanism is only activated at the x value of the current memory position. To ensure this, we place a line of bishops behind the line of white rooks, such that only one will be able to make a pin through the memory gap. We use these pins to set the control bits on transistors, so that the circuit in Figure 8 is only activated at the desired x value.

In addition to the question of geometry, there is a concern about what happens when the Turing machine does not need to change the present value in memory. Then activating the corresponding W_i variable would not give check, and there would be an undesired free move. To prevent this, we simply add a rook which is toggled between defending three different kings, controlled by the modulo 3 counter which controls the memory movement. This rook serves no purpose except guaranteeing perpetual check.

We are now prepared to construct the chess Turing machine, following basically the same procedure as in Section 5.3. The core of the machine is an array of circuits: $A_1, \ldots, A_n$; B_0, B_1, B_2; and now, M and W. All of these circuits share a single rook memory tower. A gated data flow circuit allows the outputs of these circuits to be temporarily stored in internal memory; this internal memory is wired into the rook memory tower.

Additionally, the memory tape is wired into the rook memory tower. The internal memory is wired to the movement mechanism and the writing mechanism, so that the outputs of circuits M and W can be used to move the memory gap and write to memory.

Finally, all of these mechanisms are wired to a central clock circuit. Writing to memory adds an additional move to each time step of the machine, so the clock circuit is a hexagonal path. For the sake of completeness, the steps taken in each time step of the Turing machine are as follows:

(I) A transistor is opened so black bishops at the apexes of circuits can put white kings in check, forcing white to move rooks to defend.

(II) Internal memory is reset.

(III) A gated wire is opened, copying the white rooks to another set of white rooks, which form the internal memory.

(IV) The first white rooks are reset.

(V) The memory write circuit is activated.

(VI) The black rooks in the memory movement mechanism are activated, moving the memory gap.

A sketch of the chess Turing machine is given in Figure 9.

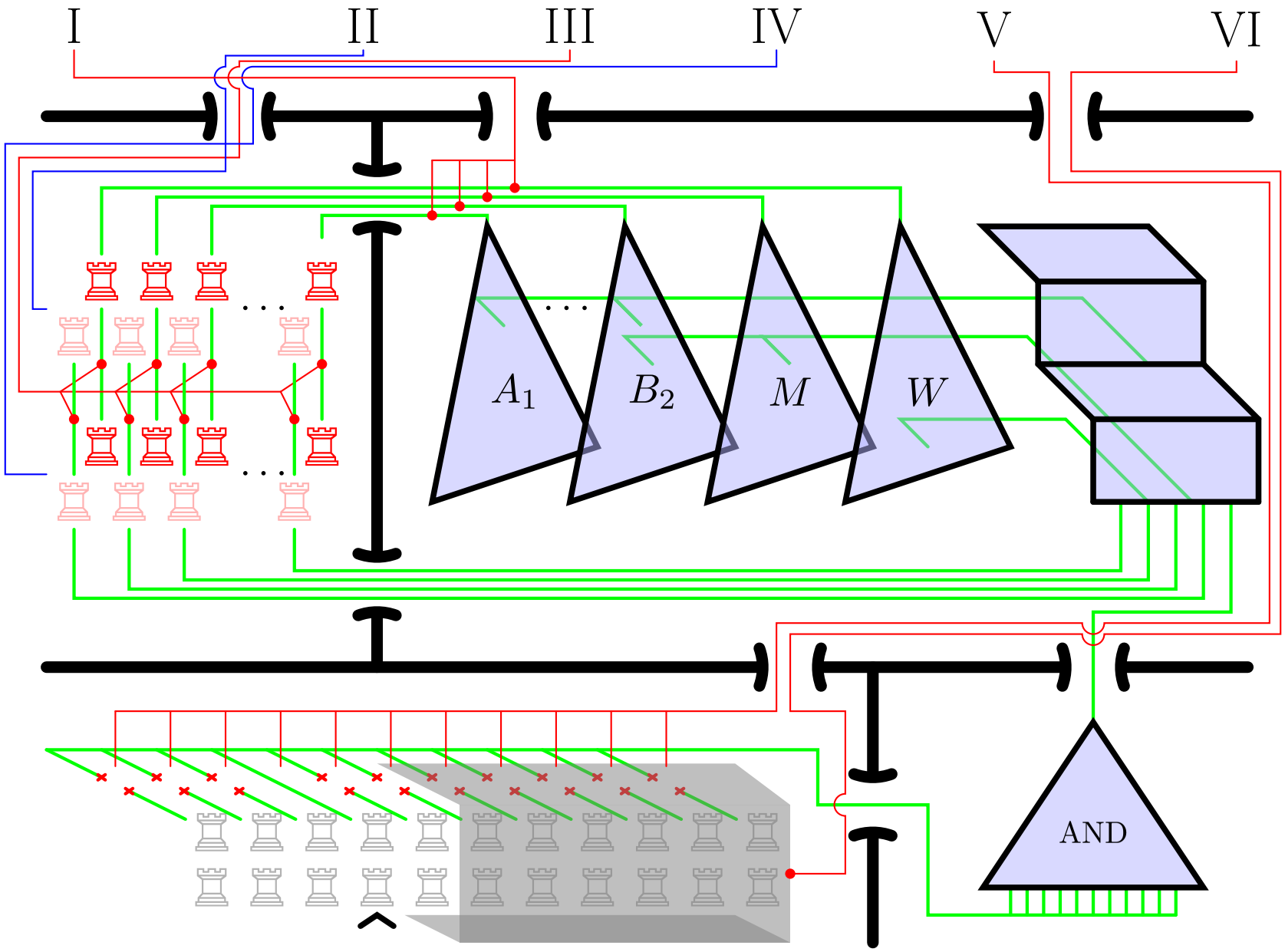


Figure 9: Schematic of a chess Turing machine. Data links are shown in green, control wires are shown in red, and reset controllers are shown in blue. Light blue triangles represent chess circuits, and the light blue staircase is the rook memory tower.

Figure 10: All hail mahogany.

7 Conclusion

Keats tells us that beauty is truth, truth beauty; in this paper, we have shown that three-dimensional chess is both truth and beauty. There can no longer be any excuse for resorting to the temptations of the lowly semimetal known as silicon. Mahogany is supreme, mahogany is lord. All hail mahogany.

References

[1] Ross Dempsey, Sydney Timmerman, and Karl Osterbauer. Chess circuits. In *Sigbovik*.