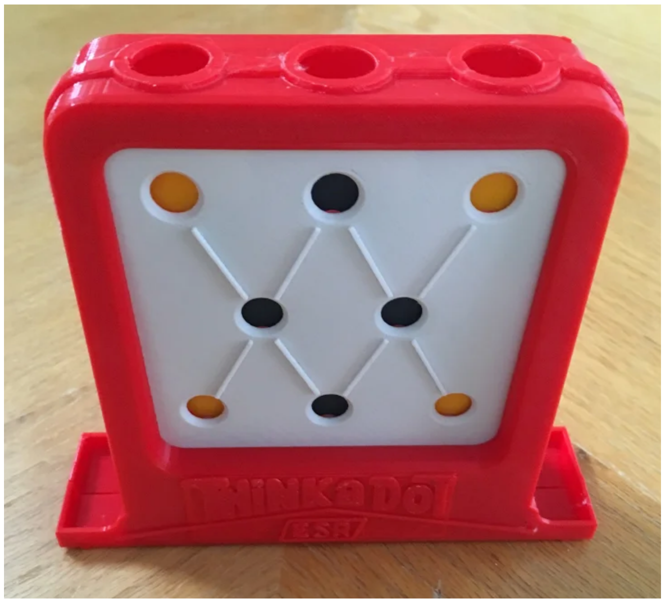
DMP Notes K. Sutner

Dropping Balls

2023/05/30 v0.8

1 A Puzzle

In the 1960s, a company by the name of *Education Science Research* marketed a mathematical puzzle, called Think-a-Dot, an invention of one J. Weisbecker. Here is a picture of the toy.



Perhaps surprisingly, the toy can be used to introduce and motivate a whole number of concepts in discrete math, and in particular concepts that are relevant to computer science. This observation was not lost on some members of the MIT faculty, who used the game in their introductory theory of computation course half a century ago.

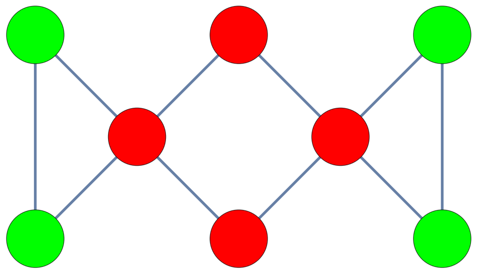
These notes present a fairly detailed study of Think-a-Dot, using only rudimentary tools in sections 2 and 3. Section 4 suggests some challenge problems that push a bit further. Lastly, section **??** introduces a more advanced attack, and proposes rather open-ended and difficult questions[†]. It's safe to skip this part for the time being unless you have an exceptionally strong math background.

[†]I do not know all the answers myself. Extra Credit.

2 The Right Questions

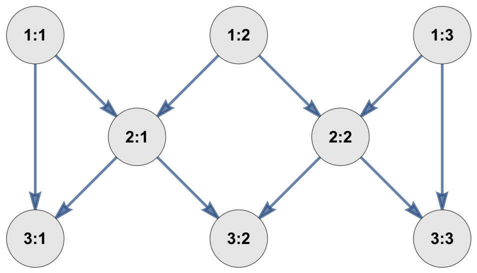
From the picture, one can see more or less that there are 8 locations in the device, organized in a trellis structure, each colored yellow or blue. Moreover, there are three "slots" at the top. The idea is that one can drop a ball into one of these slots, which will then fall through some of the 8 locations, following the lines in the picture, until it drops out at the bottom. What makes the device interesting is that, internally, in each of the locations there is a little gizmo, a flip-flop that directs the ball either left or right. The flip-flops have memory, so if the last ball has gone to the left, the next one will be directed to the right, and vice-versa. The state of each flip-flop is indicated by the yellow and blue displays. Initially all flip-flops are in state left. The goal is to insert a sequence of balls at the top so that a particular yellow/blue pattern appears.

To improve visibility, we will use colors green and red instead (for left and right, respectively), so the state of the toy in the picture abstractly looks like so:



It is a good exercise to figure out how to get there starting from all green; 7 balls suffice.

Whenever we to refer to particular flip-flops we will use the following naming convention, based on matrix-like rows and columns.



The slots where balls can be inserted correspond to 1:i for $i = 1, 2, 3$. All the other locations can only be reached indirectly, which makes the puzzle interesting.

Question: What are the right questions?

Let's be a bit more formal in our description of the trellis automaton so that we can state a few questions with clarity. At any point, the state of the device is determined by the collection of the orientation of all the flip-flops. To lighten notation, let use bits rather than colors or geometric directions, say, 0 for left/green and 1 for right/red. So the whole state of the device is given by a list of bits of length 8[†].

[†]A hacker might want to think of a byte.

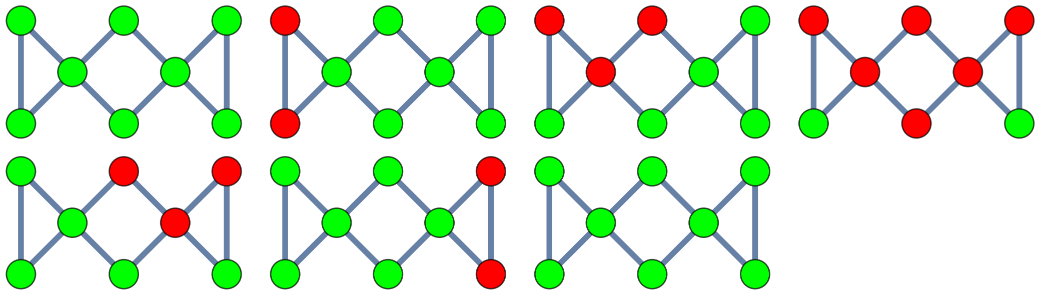
Let's write the basic atomic actions of dropping a ball into the left, middle, right slot on top as a, b and c, respectively. From the description of the device, given a state X and an atomic action s, the device goes into a uniquely determined new state Y. We write

$$\widehat{\alpha}(X, s) = Y$$

for the result of a single ball drop in slot s. A single ball is boring, we will be concerned with whole sequences of ball drops. For example, the sequence abb would produce the result $\widehat{\alpha}(\widehat{\alpha}(\widehat{\alpha}(X, a), b), b)$, which we will write compactly as $\alpha(X, abb)$. Technically, we can define a corresponding action function α by recursion as follows:

$$\begin{aligned} \alpha(X, \varepsilon) &= X \\ \alpha(X, s_1 \dots s_{k+1}) &= \widehat{\alpha}(\alpha(X, s_1 \dots s_k), s_{k+1}) \end{aligned}$$

Here we have written ε for the empty sequence of drops, which obviously does not affect the state at all. This is fairly close to how this definition would be expressed in a suitable programming language. Note that $\widehat{\alpha}$ is just a finite lookup table, but α requires some computation (well, it's just repeated table lookup, but still). Here is an example. Write $\mathbf{0}$ for the all-zeros state. Then the result of executing $\alpha(\mathbf{0}, abcabc)$ in steps looks like so.



Make sure to check the details in this picture carefully. Also note the rather surprising result: in the end, we are back to where we started.

Here are some questions that arise fairly naturally.

- Reachability

  Given two states X and Y, is there an action sequence u such that $\alpha(X, u) = Y$? There are actually two parts to this: first, how do we check if u exists, and, second, how do we compute such a sequence, if indeed one exists?

- Transfer Sequences

  If Y is reachable from X, what is the shortest sequence u such that $\alpha(X, u) = Y$? How do we compute it?

- Reversibility

  If u takes X to Y, is it always possible to find another sequence v that takes us back to X: $\alpha(X, u) = Y$ and $\alpha(Y, v) = X$. If not always, when is it possible to find a suitable v?

- Classifying Actions

  Two different action sequences may have the same effect in the sense that for all states X we always have $\alpha(X, u) = \alpha(X, v)$. In this case we call u and v equivalent. What can one say about equivalence? How do we determine whether two sequences are equivalent?

Since there are only $2^8 = 256$ possible states, we could certainly solve the reachability problem by brute-force: given X, systematically enumerate all reachable states by applying the atomic actions over and over again, until no new states pop up any more. Properly organized, this will also solve the transfer sequence problem. Implementing a brute-force attack in a programming language of your choice is a nice exercise, and highly recommended. Still, we are ultimately interested in insights, we would like to understand why the program produces certain answers. Incidentally, better structural understanding often produces elegant and fast algorithms as a side-effect (but it's perfectly fine to start with brute-force)[†]. Imagine we had a trellis with 100 flip-flops; we really need some theory to handle this sort of problem.

Let's ignore reversibility for the time being, we'll come back to this problem later.

With respect to the classification question, note that there must be equivalent action sequences, and in fact infinitely many equivalent sequences. To see why, observe that any action sequence s produces a map from states to states:

$$\begin{array}{rccc} \widehat{s}: & \mathbf{2}^8 & \longrightarrow & \mathbf{2}^8 \\ & X & \longmapsto & \alpha(X, s) \end{array}$$

There are only finitely many such maps (though admittedly a great many: $256^{256} \approx 3.23 \times 10^{616}$). On the other hand, there are infinitely many action sequences, so lots of them must be equivalent, $\widehat{t} = \widehat{s}$. But finding an equivalent sequence for a particular s is a different problem: we don't know whether there is an equivalent sequence for s at all and even if there is some suitable t, we do not know how long it might be. Never mind that there are exponentially many sequences of length n, we are going to run into serious computational efficiency problems. We really need to develop some understanding of equivalence.

Let me harp on this some more: experimental computation is your friend and can help tremendously to build up intuition and understanding. A little quick-and-dirty program often produces better data than anything anyone could do by hand. Later, when one has better understanding of the problem in question, one can refine the program to produce better data, generate conjectures, find examples and counterexamples, and so on. An interpreted language with a nice user interface is best to this sort of interactive computation, modern computer algebra systems in particular work very well. For instance, the pictures in these notes are all generated by *Mathematica*. CMU has site license, try it out.

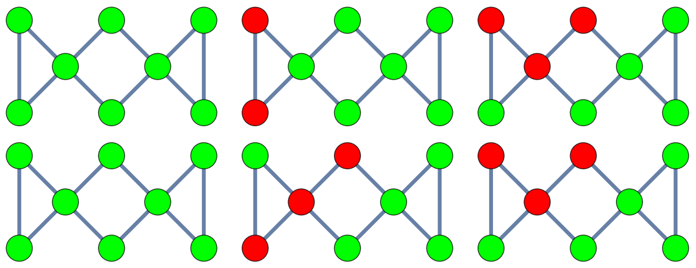
3 Basic Answers

First let's tackle reachability: when is there an action sequence u that takes a given source state X to a given target state Y?

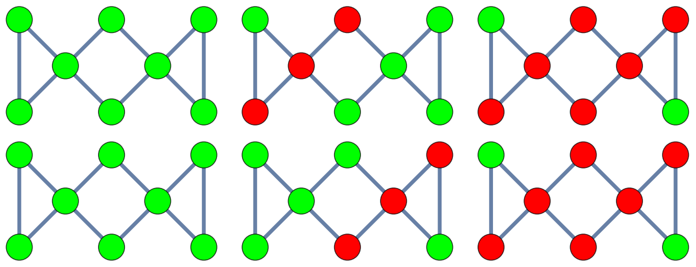
[†]As D. Knuth likes to say about programming: *Premature optimization is the root of all evil.*

Note that any ball dropped into the device will flip a bit in the top row, and another in the bottom row (though not necessarily in the middle row). Let us refer to the parity of a state as the sum of all 1-bits in the top and bottom rows, counting modulo 2. It follows that any action sequence preserves parity. So any state of different parity cannot possibly be reachable and we must be dealing with at least 2 types of states.

Great, but is that all? Intuition seems to be of little immediate help here, so a little experimentation is indicated. The most obvious approach would seem to try out very short action sequences. Single atomic actions are not particularly helpful, so how about length 2, say aa, ab, ba and so on. For example, here is result of action ab (top row) and ba (bottom row) on state $\mathbf{0}$:



The intermediate results are different, but the final state is the same. This could well be coincidence, but here is the same experiment for bc and cb.



Moreover, the analogous experiment for ac and ca produces a similar result. So we have $\alpha(X, ab) = \alpha(X, ba)$, $\alpha(X, ac) = \alpha(X, ca)$ and $\alpha(X, bc) = \alpha(X, cb)$. If you don't trust the program, this can easily be verified by hand[†]. It now follows by induction that for starting state $\mathbf{0}$ only the number of actions a, b and c matters, but not their order. Make sure to carry out the induction.

Below is a slightly more general argument that has the advantage that it generalizes to more complicated trellises, see section **??**.

Lemma 3.1 (Commutativity)

For all states X and sequences u and v, we have $\alpha(X, uv) = \alpha(X, vu)$.

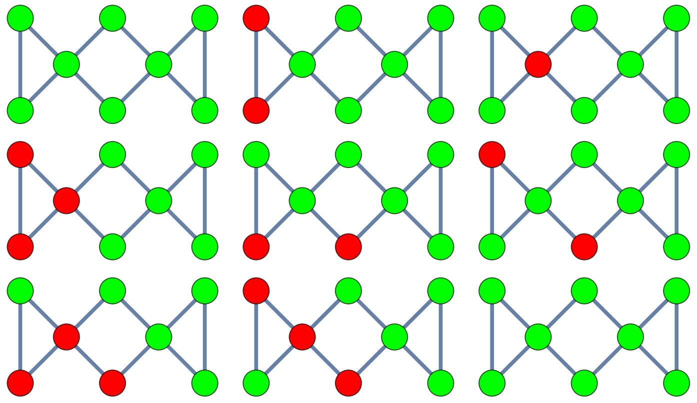
[†]Incidentally, checking the results of experimental computation is hugely important; it is easy to slip up somewhere, get misleading results, and then waste a lot of time following the wrong path before everything collapses.

Proof. We will only show $\alpha(X, ab) = \alpha(X, ba)$ and leave the other two combinations of atomic actions to the reader. By induction, it suffices to handle these basic cases.

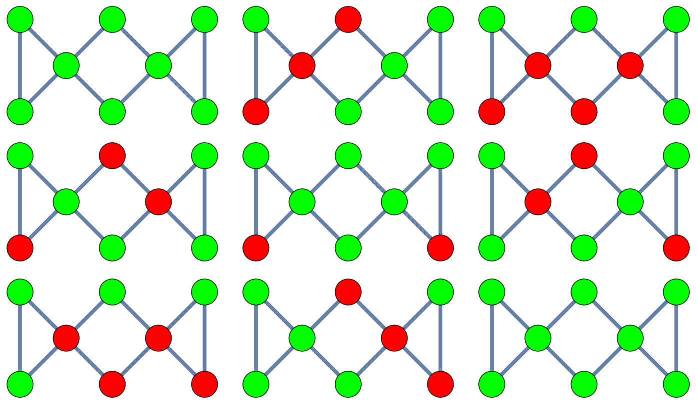
It is clear that our claim holds for all the bits of X in the top row. If the paths of the two balls intersect at most in the bottom row, we are done. Otherwise they intersect in 2:1. By the definition of a flip-flop, the two balls arriving there will ultimately leave the flip-flop in the original state, and will send one ball each to 3:1 and 3:2 (albeit different balls). Hence the changes in the bottom row are also the same.

□

Commutativity simplifies matters a bit. We write u^k for k repetitions of sequence u, $k \geq 0$. By the lemma, we only need to consider sequences of the form $a^k b^\ell c^m$ where $k, \ell, m \geq 0$. So the next step should be to study sequences of just a single atomic action. After all, $\alpha(X, a^k b^\ell c^m) = \alpha(\alpha(\alpha(X, a^k), b^\ell), c^m)$. Let's start with action a. There is a huge surprise: after 8 steps we are back to where we started.



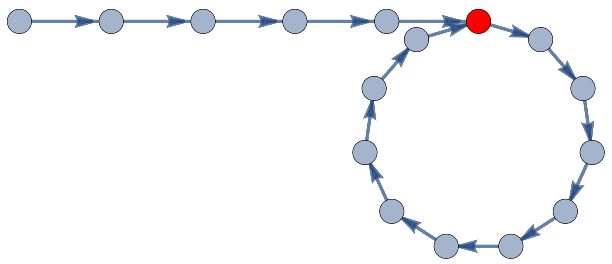
Again, this could be coincidence: this might not work for other initial states, and/or for other atomic actions. Well, here is the analogous picture for action b.



There are really two separate questions, here, let's try to pull them apart. First off, if our observations pan out, our game must be reversible: we can undo any sequence of actions by applying a suitable "opposite" sequence. For example, action a can be undone by a^7, and similarly for the others. This suffices by commutativity. To establish reversibility, we first need to show that for any atomic action s and any states X and Y we have

$$\alpha(X, s) = \alpha(Y, s) \quad \text{implies} \quad X = Y$$

This is a property called injectivity. What does injectivity have to do with reversibility? Since there are only finitely many states, applying action s over and over again must ultimately lead to a repetition, things must wrap around as in the following picture (where each arrow represents one application of action s).



But then injectivity is broken at the red point, and the only way to avoid this is to wrap around at the beginning, We wind up with a simple loop, and we can get back to the origin by following forward edges. Here is the argument for injectivity.

Lemma 3.2 (Injectivity)

Let s be an atomic action, X and Y two states such that $\alpha(X, s) = \alpha(Y, s)$. Then $X = Y$.

Proof. Consider the paths traced by the ball in the computation of $\alpha(X, s)$ and $\alpha(Y, s)$, respectively. Both paths start at the same place; if they agree entirely, we are done (why?). So suppose they first branch at location p. But then $X(p) \neq Y(p)$, and that discrepancy remains after both actions are performed: a ball can never fall upwards. This contradicts our assumption.

□

The question remains how long these loops are for each of the atomic actions. From our computations we know that the answer is 8, and the intrepid can check this out by hand.

Proposition 3.1 (Loops)

For any atomic action s and any state X, we have $\alpha(X, s^8) = X$.

Still, it would be nice to have some sort of explanation as to why the answer is 8. To this end, define the light cone of any slot to be the collection of flip-flops that can be reached by a ball dropped into the slot. For example, the light cone of 1:1 consists of 1:1, 2:1, 3:1 and 3:2. First consider the source state $\mathbf{0}$, and focus on the diagonal 1:1, 2:1, 3:2. Thinking of these 3 bits as a number written in binary, with least-significant digit first, we can see that action a implements a binary counter.

The light cone of a has 16 possible states, and we have just encountered 8 of them. The missing ones are obtained by flipping the bit in 3:1. Hence $\alpha(X, s^8) = X$ for all states X: the bits outside the light cone do not matter.

The argument for c is quite similar, except that this time the counter decrements. For atomic action b we are dealing with a forward and a backward counter.

As an immediate consequence, we have $\alpha(X, u^8) = X$ for any actions sequence u. Now recall that two action sequences u and v are equivalent if $\alpha(X, u) = \alpha(X, v)$ for all states X, we will write $u \equiv v$ for equivalence. By our preceding results, we now know that any sequence is equivalent to one in normal form

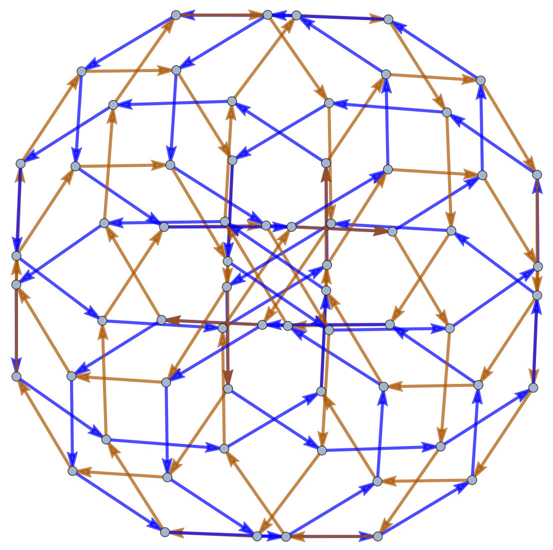
$$a^k b^\ell c^m \qquad \text{where} \qquad 0 \leq k, \ell, m < 8$$

Hence, up to equivalence, there are at most $8^3 = 512$ distinct sequences. This is really quite amazing, we can now cut down infinitely many sequences to just 512. As a result, brute-force computation may actually answer questions that intuitively seem to be out of scope. This interplay between computation and theory is not uncommon.

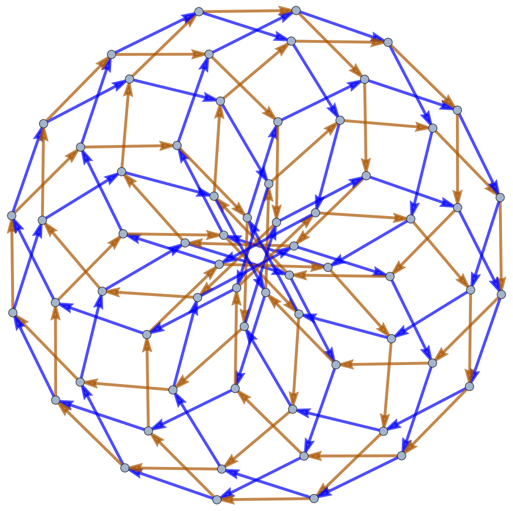
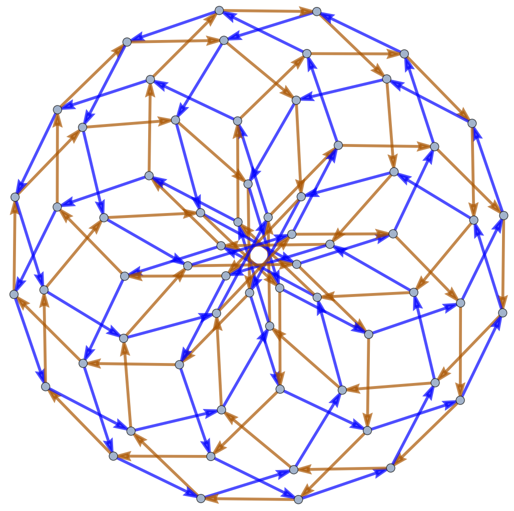
Normal form is also convenient to express the reverse action sequence: If $\alpha(X, a^k b^\ell c^m) = Y$, then $\alpha(Y, a^{-k} b^{-\ell} c^{-m}) = X$ where the exponents are supposed to be computed modulo 8. This is certainly not clear from the original description of the game.

One might wonder if there is a way to visualize the properties of our action, rather than just explaining them in algebraic terms. The obvious tool is to draw a reachability graph: the vertices are the states, and there is an edge from X to Y if some atomic action takes X to Y. Unfortunately, these pictures get a bit messy; there are too many states and arrows. Here is a subgraph, showing just one cluster of size 64, obtained by using only actions a and b. Cluster here simply means that any two states in that collection of states are mutually reachable. There are 3 others, that are all isomorphic to this one.

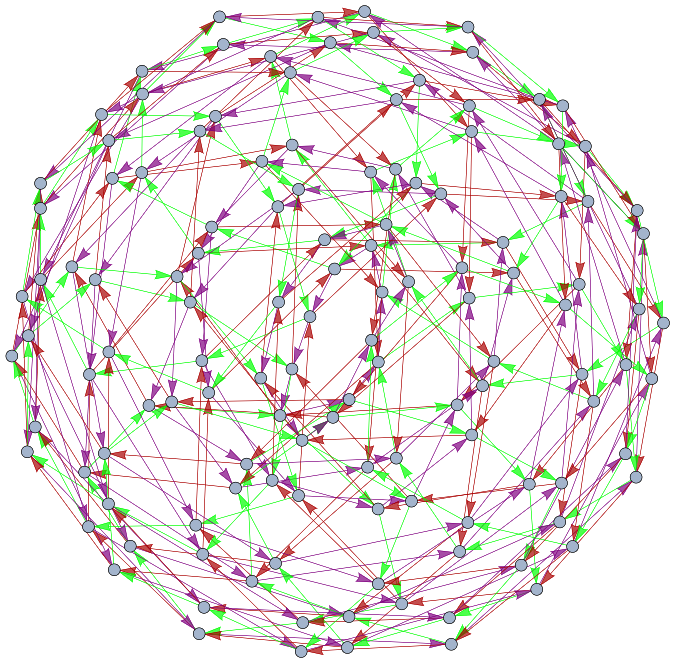
With a bit of effort (and by blowing up the pictures), one can see blue/orange squares on the "surface" that indicate that we can interchange a and b. One can also more or less see the darker blue edges arranged in cycles of length 8, another confirmation of proposition 3.1.



The corresponding pictures for combinations a, c and b, c look similar.

Using all three actions we get two heavily tangled balls of yarn of size 128 each, here is one of them.



It may seem that we have a fairly complete understanding of the game, but there is still work to do: we do not yet know how to deal with transfer sequences, shortest action sequences that transform state X to state Y. From our normal form $a^k b^\ell c^m$, $0 \leq k, \ell, m < 8$, it follows immediately that at most 21 atomic actions suffice. But perhaps we can find other identities $a^k b^\ell c^m \equiv \varepsilon$, that would allow us to further reduce the number of actions needed. By parity, all exponents have to be even for that to happen, so it is natural that exponents 2 might pop up. One can easily check that shorter sequences such as a^2 or $a^2 b^2$ do not work, but then a minor miracle occurs.

Lemma 3.3 (Identity)
The action sequence $a^2b^2c^2$ is equivalent to ε.

The proof of this lemma is based on a careful and slightly tedious examination of what it means for a state to be a fixed point under $a^2b^2c^2$. We collect all the necessary information in the next proposition, and leave it to the dear reader to check that the lemma indeed follows. To keep notation manageable, we write X_{ij} for the bit in position $i{:}j$ in state X. More importantly, we give up on α and simply write Xu instead of $\alpha(X,u)$. This is ultimately justified because an action is indeed some sort of multiplication, and so it's tempting to use similar notation. It will always be clear from context what exactly we mean.

Proposition 3.2 *The following identities hold for all states Z.*

1. $Z_{1j} = Zaa_{1j} = Zbb_{1j} = Zcc_{1j}$.
2. $Z_{21} \neq Zaa_{21}, Zbb_{21}$ *and* $Z_{21} = Zcc_{21}$; $Z_{22} \neq Zbb_{21}, Zcc_{21}$ *and* $Z_{22} = Zaa_{22}$.
3. *For the third row we have*

$$\begin{aligned}
Z_{31} = Zaa_{31} &\iff Z_{21} = 0\\
Z_{31} = Zbb_{31} &\iff Z_{21} = 1\\
Z_{32} = Zaa_{32} &\iff Z_{21} = 0\\
Z_{32} = Zbb_{32} &\iff Z_{21} \neq Z_{22}\\
Z_{32} = Zcc_{32} &\iff Z_{22} = 1\\
Z_{33} = Zbb_{33} &\iff Z_{22} = 0\\
Z_{33} = Zcc_{33} &\iff Z_{22} = 1
\end{aligned}$$

This looks much worse than it is; all the claims can be proven by a bit of diagram chasing.

By the lemma, there are other, boring equivalences: $a^{2k}b^{2k}c^{2k} \equiv \varepsilon$. A harder question is whether we missed any substantially different identities. To show that this is not the case, we first exploit lemma 3.3 some more. We can think of our equivalences as rewrite rules that allow us to simplify action sequences. In this context, it is convenient to interpret an action sequence simply as a word, a string over the alphabet $\Sigma = \{a, b, c\}$. We then use word-processing to manipulate these strings without changing their meaning as action sequences. Our goal is simplification: define the weight of a word $a^k b^\ell c^m$ to be $k + \ell + m$; we want our rules to reduce weight. Note that this forces the simplification process to stop after finitely many steps, we certainly cannot keep going forever.

Proposition 3.3 *Let r, s, t be any permutation of a, b, c. The following rewrite rules on action sequences produce equivalent sequences.*

$$\begin{aligned}
a^2b^2c^2 &\rightsquigarrow \varepsilon\\
r^4s^4 &\rightsquigarrow t^4\\
r^6 &\rightsquigarrow s^2t^2
\end{aligned}$$

For example, we have

$$a^7bc^4 \rightsquigarrow ab^3c^6 \rightsquigarrow a^3b^5$$

A word like a^3b^5 that cannot be further simplified by any of the rules is called irreducible. How many irreducible words are there? Let $r^k s^\ell t^m$ be irreducible, and assume $k \leq \ell \leq m$. Then $k \leq 1$, $\ell \leq 3$ and $m \leq 5$, but it is not exactly clear how many words satisfy these constraints. One can write a little program to determine the number of irreducibles to be 128. Or we can use combinatorial counting, in this case the infamous inclusion/exclusion principle, to determine that the number of irreducibles is

$$288 - 312 + 256 - 144 + 48 - 8 = 128$$

This is quite tedious to do by hand, but a computer algebra system can perform these calculations easily.

The result 128 seems to confirm our computational results: there are exactly two clusters of states, with even and odd parity, respectively. But there is still one lingering problem: there is no obvious reason why two inequivalent sequences u and v could not produce $Xu = Xv$ for some state X. This would wreck the size of our clusters. As it turns out, this actually cannot happen: once we know that $Xu = Xv$ for any state X, we can already conclude equivalence. Note that this is far from obvious, on the face of it there is no reason why there could not be another state Y such that $\alpha(Y, u) \neq \alpha(Y, v)$. To see why this cannot happen, let us briefly detour to an algorithmic question: given states X and Y, how can we test whether Y is reachable from X, and, if so, how do we compute a corresponding action sequence?

We can extract an algorithm from the observations about binary counters above: by dropping a suitable number of balls into slots c, then b, then a we can generate almost all states. More precisely, to get to Y from X, do the following:

- Let $m = X_{13} + Y_{13} \bmod 2$.
- Choose ℓ such that $Xb^\ell c^m$ agrees with Y on 1:2, 1:3, 2:2 and 3:3.
- Choose k so that $Xa^k b^\ell c^m$ agrees with Y everywhere except possibly on 3:1.

Now consider the map f from states to states that flips the bit in position 3:1. Clearly f is a bijection, and $f(f(X)) = X$. According to the algorithm, we can either reach Y from X, or we can reach $f(Y)$. So there are indeed exactly 2 clusters, distinguished solely by parity. We can now establish our last result.

Lemma 3.4 (Freeness)

Suppose $Xu = Xv$ for some state X. Then $Zu = Zv$ for all states Z.

Proof.

Half of the argument is easy: if Z is in the same cluster as X, say, $Z = Xw$, we have by commutativity:

$$Zu = Xwu = Xuw = Xvw = Xwv = Zv$$

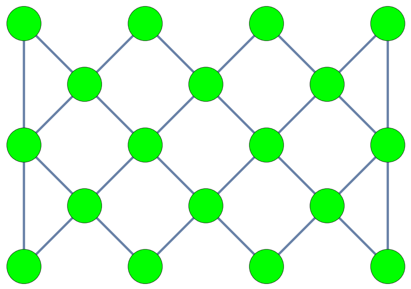
For the states from the other cluster, use the map f from the preceding remark. One can check that $f(X)u = f(Xu)$, so the result carries over to the second cluster.

$\square$

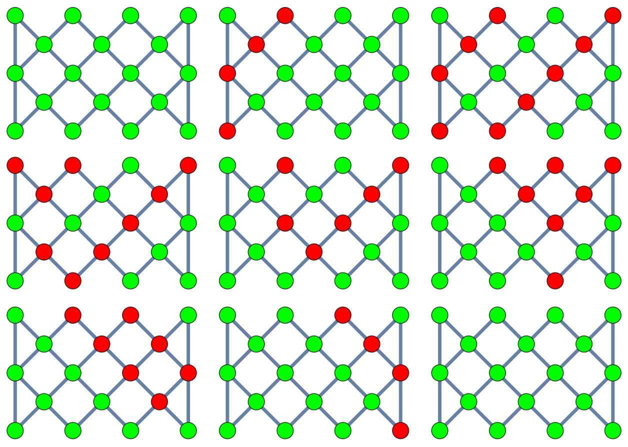
With a view towards generalizations in the next section, the last argument is quite fragile. It is a good exercise to try to make the argument more robust by avoiding any reference to the global behavior of our action and focus instead the details of how the action works on a state.

4 A Challenge

Looking at the trellis pictures from above, our contraption is built from two copies of the basic ╳-shaped component, we have been dealing with a 1×2 trellis. Well, how about this 2×3 trellis?

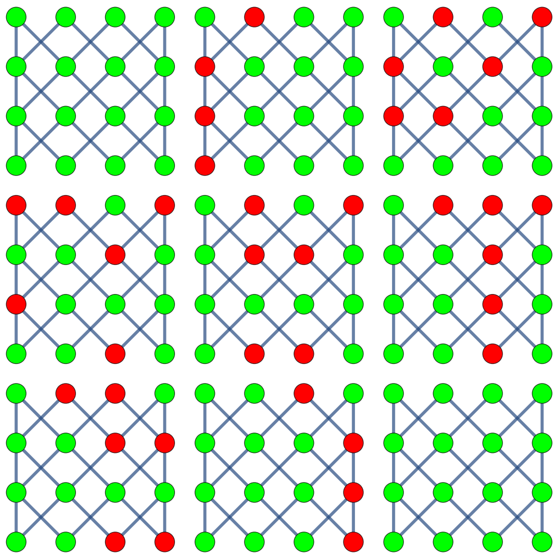


What stays the same, what changes? Try to reuse as much information as ever possible, both for general strategy and for proofs. As a teaser, here is the result of the "random" action $w = bdaacdbc$ on $\mathbf{0}$ for this larger trellis.



Can you see why this follows from our previous results?

Other than changing the size, we could also adjust the topology of the trellis. For example, we could keep the number of flip-flops constant in each row, as shown in the next picture. One could argue that this is actually a slightly less complicated scenario. We would expect that these uniform trellises behave in a similar manner to the original ones. Still, it is not clear which of our results carry over, and which break. Here is a case that is quite manageable because of symmetries: a 4×4 plain trellis. The picture shows the orbit of $\mathbf{0}$, again under $w = bdaacdbc$.



Could this be coincidence? Any conjectures?

Needless to say, there are arbitrary $n \times m$ grids to deal with, both with the original layout and the simplified version.