Demos

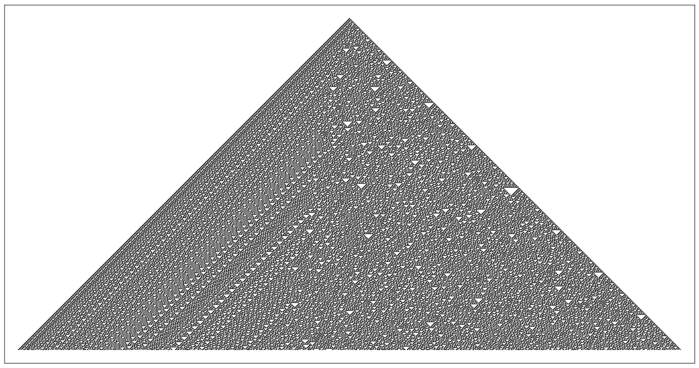
Demo 1: Cellular Automata
A special kind of Turing Machine

Cellular Automata

Let’s talk about cellular automata, because they are a nice system to think about computation.

```
In[•]:= CellularAutomaton[30, {{1}, 0}, 500] // ArrayPlot
```

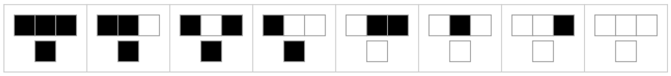
Out[•]=



Here is the rule table for Rule 30:

```
In[•]:= RulePlot[CellularAutomaton[240]]
```

Out[•]=



It means that for each cell, it looks at its own state (black or white) and the states of its two adjacent neighbors, and depending on what all those states are, the cell either changes its color or not.
This is what the top three cells are, a center cell and its two neighbors. The cell underneath the three is what will happen to the cell in the next time step for this “neighborhood” configuration.
Notice how the rule table gives the outcome for all possible neighborhood scenarios.

Question 1

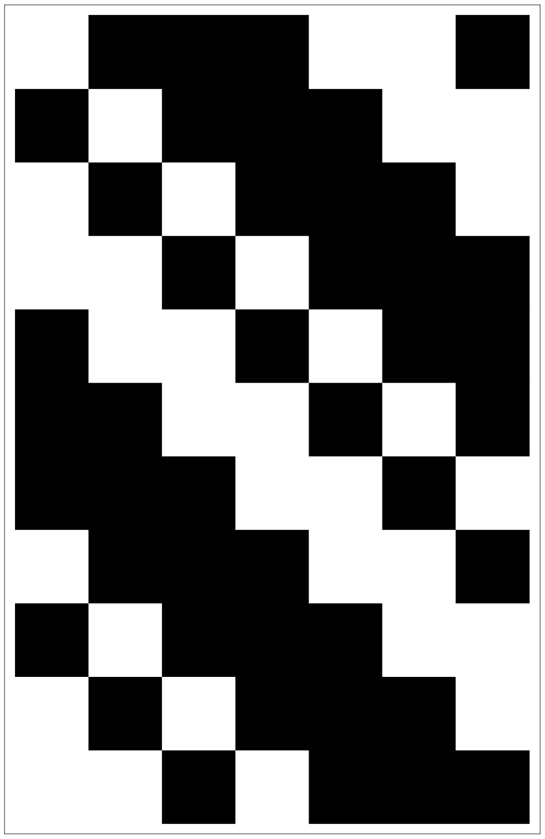
By only looking at the rule table for a cellular automata, can you determine if the cellular automata will produce something “interesting” or eventually reach a stopping point?

Answer 1

No :(This is the halting problem. To see what kind of patterns these rules make in a cellular automata, you actually have to run them. Check it out, for the rule table above:

In[•]:=
```
CellularAutomaton[240, {0, 1, 1, 1, 0, 0, 1}, 10] // ArrayPlot
```

Out[•]=

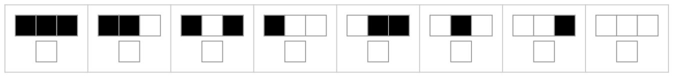


Question 2

What about this rule table? **Can you imagine what patterns this CA will produce just by looking at the rule table?**

In[•]:=
```
RulePlot[CellularAutomaton[0]]
```

Out[•]=

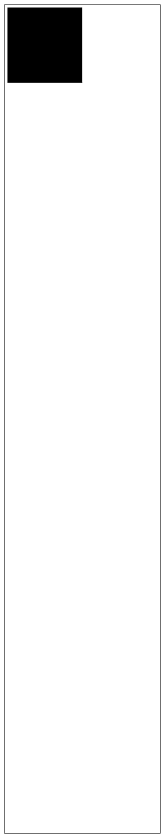


Answer 2

Well, no matter what a cell and its neighbors look like, it will always make a white cell at the next time step. So any input will give a bunch of white cells.

```
In[]:= CellularAutomaton[0, {1, 0}, 10] // ArrayPlot
```

Out[]=



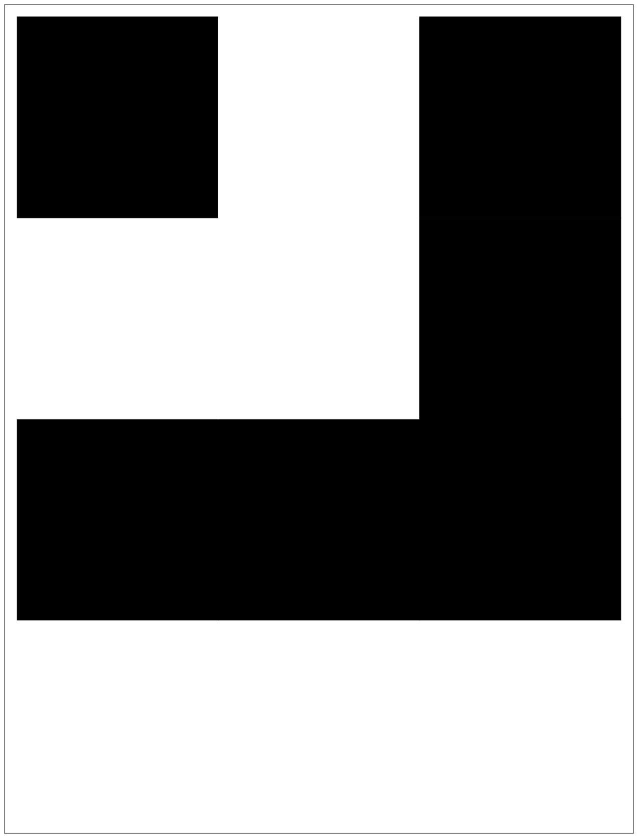
Question 3

Wait, I thought we weren’t able to guess that because of the halting problem! Actually, we are able to make some approximations. The halting problem is a bit more general than that. There’s no way to write a program that will tell us when ***any*** cellular automata rule table will produce a pattern that will stop or make something interesting forever.

However, with cellular automata, we already know the rule that created outputs, which is cool, but not anything like real data we get from the real world. For example, what rules are creating Tweets? What rule is driving diseases and cancer? Is there any way to know? Let’s think about it in cellular automata land.

Here’s some data produced from a cellular automata. We don’t know the rule. **On pen and paper, can you try to reproduce the rule table based on what you see here? Remember that when you run into the left or right edge, the boundary just wraps around (the CA lives on a cylinder and this view how the cylinder is cut and laid flat).**

Out[•]=



Answer 3

We see from the first to the second row:

{1,0,1}->0

{0,1,0}->1

{1,1,0}->0

The second to the third:

{0,0,1}->1

{0,1,0}->1

{1,0,0}->1

And the third to the fourth:

{1,1,1}->0

{1,1,1}->0

{1,1,1}->0

We need to see the outcomes for all 8 possible neighborhoods to be ABSOLUTELY sure of the rule that made this output. This shows outcomes for neighborhoods:

{1,1,1}

{1,1,0}

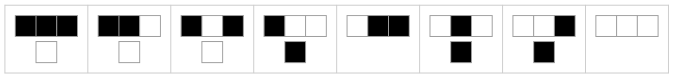
{1,0,0}

{1,0,1}

{0,1,0}

{0,0,1}

We only have 6/8 so we cannot say for sure which rule made this.



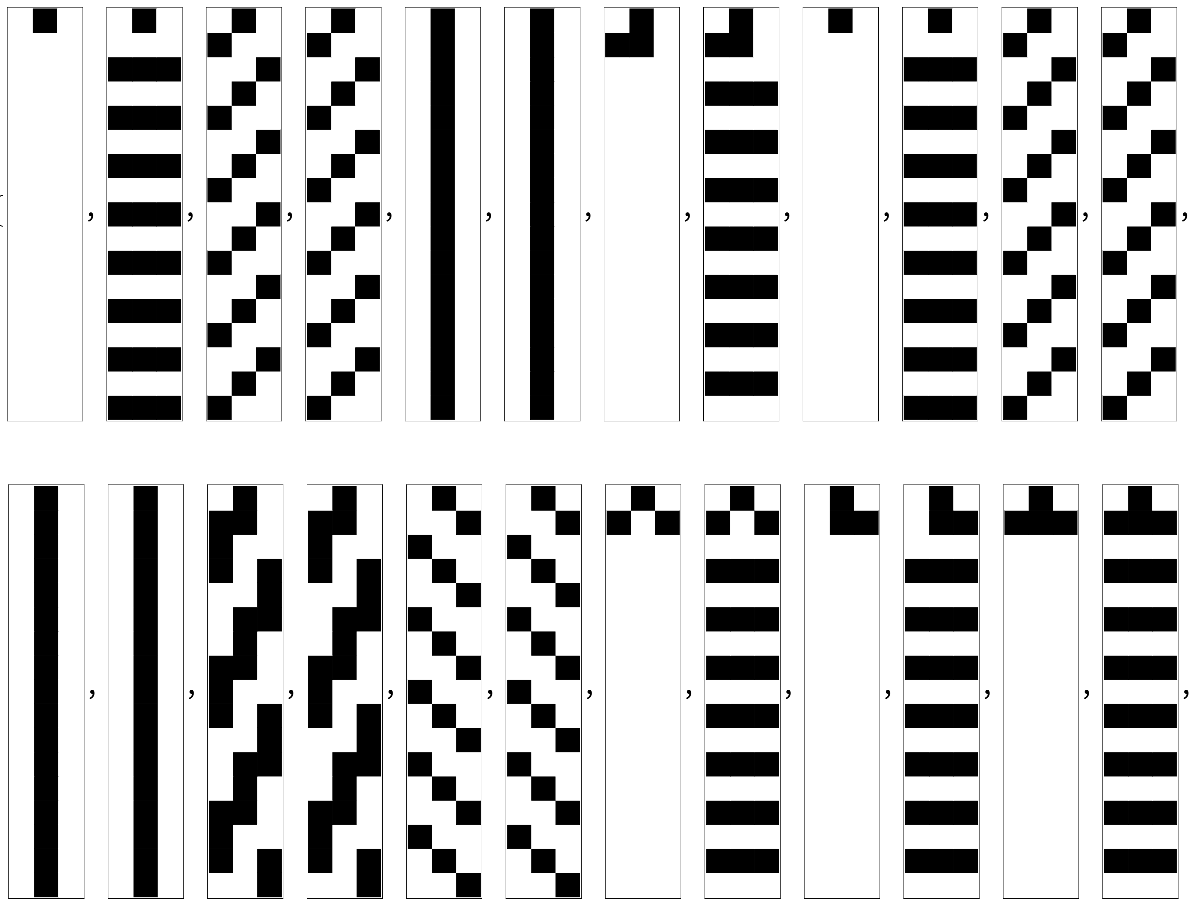
Question 4: Bonus!

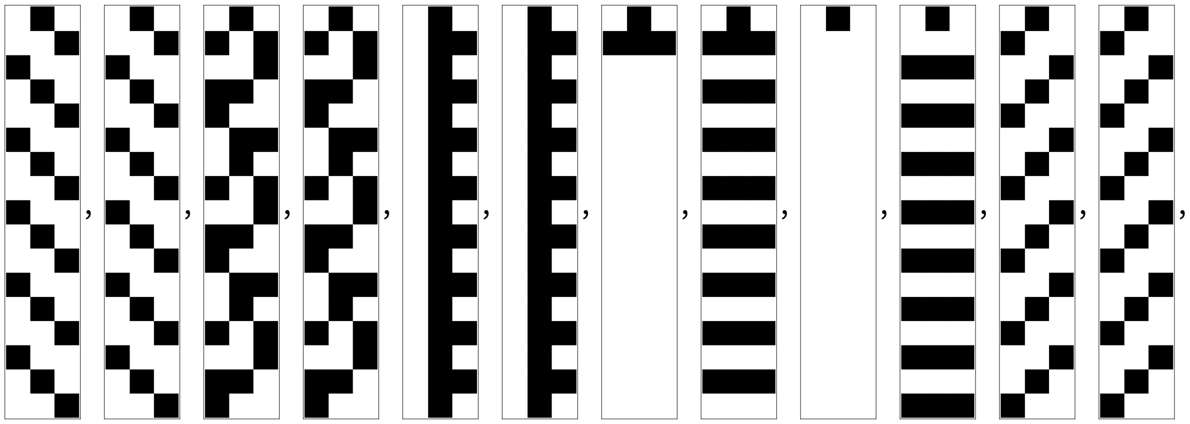
Is it possible to have a cellular automaton that continuously expresses all possible states over and over again? Assume the size of the cellular automata is 3 cells wide, like shown in this example:

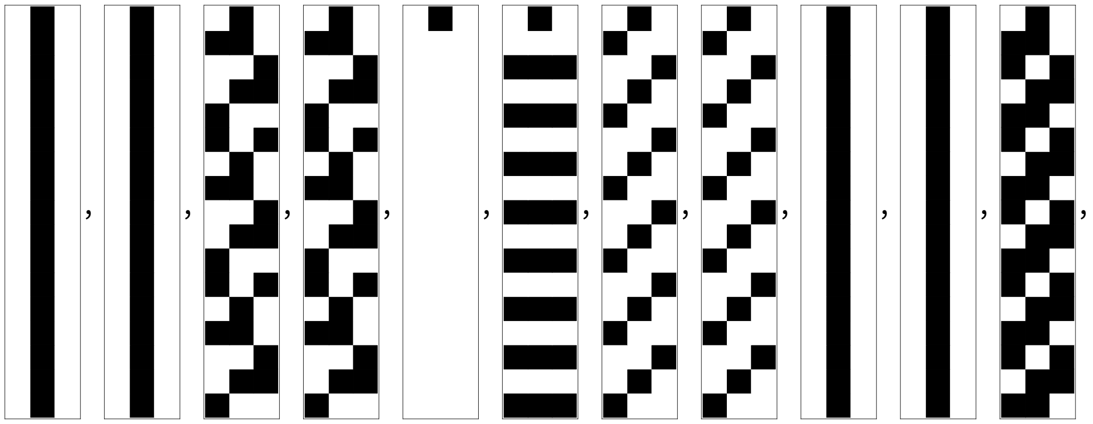
Hint: How many possible states are in a cellular automata of width 3? (2^3 = 8)

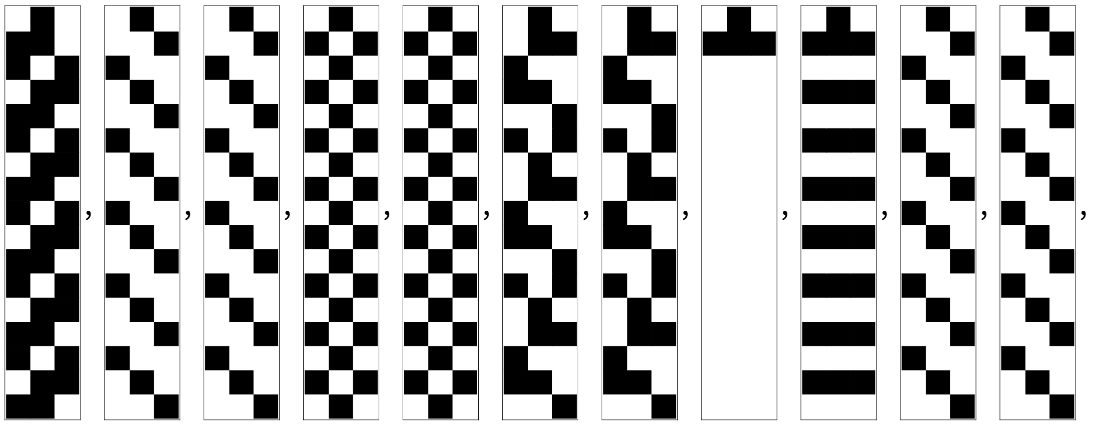
```
In[]:= Table[CellularAutomaton[i, {0, 1, 0}, 16] // ArrayPlot, {i, 0, 255}]
```

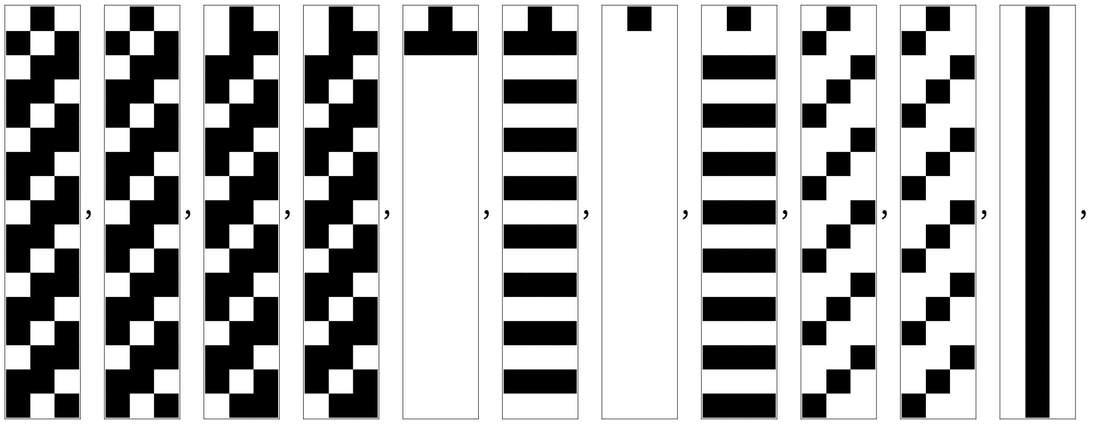
Out[]= {

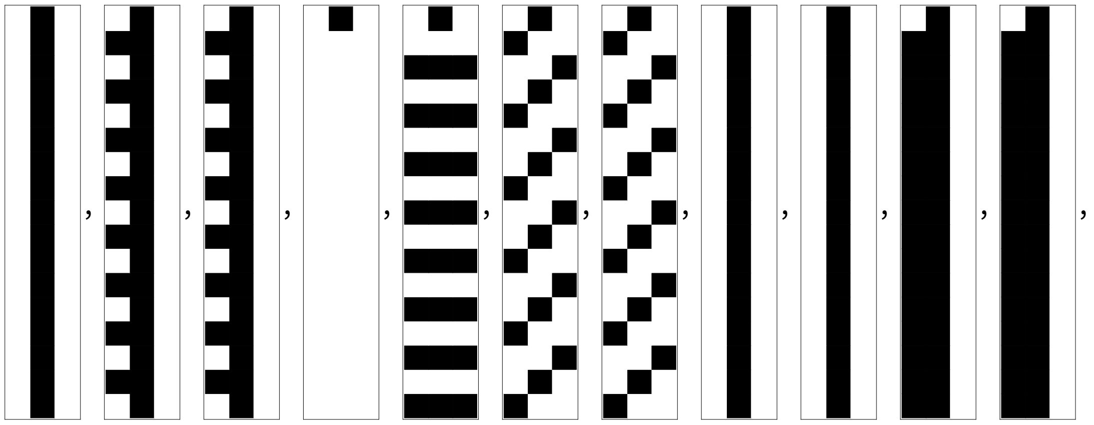


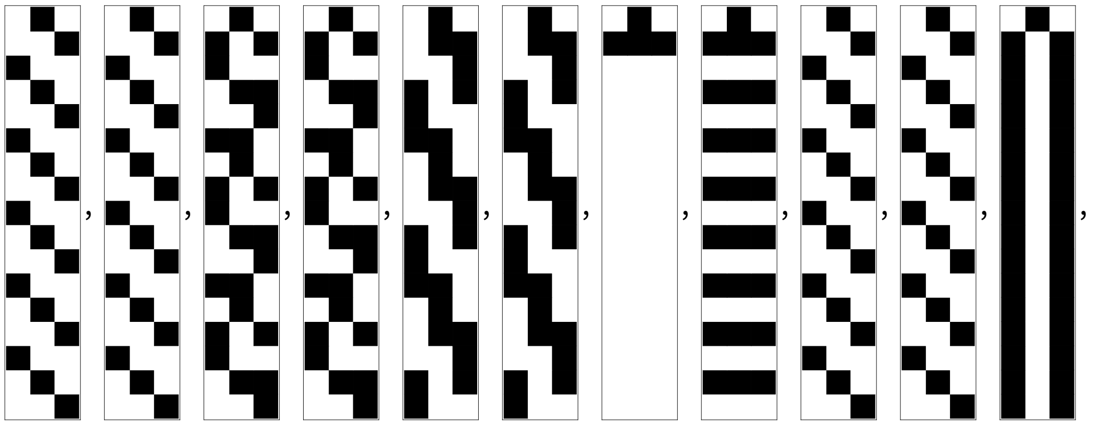


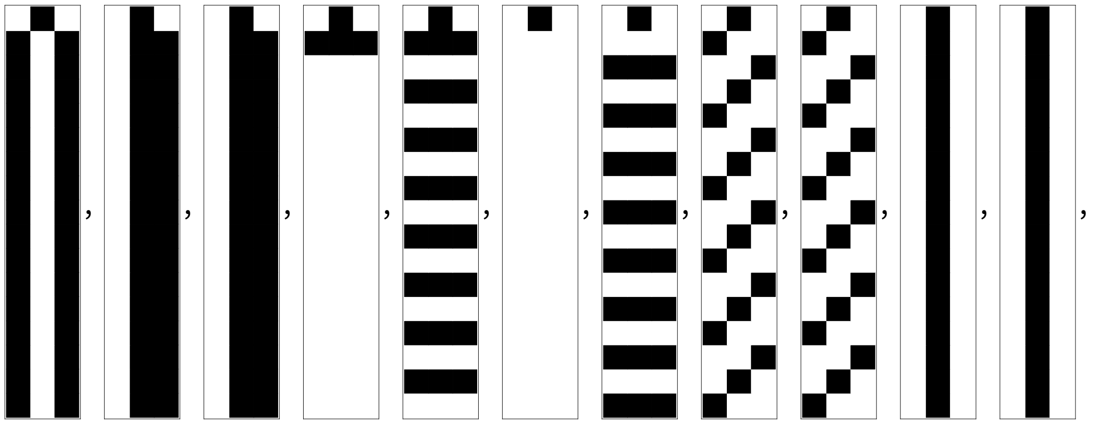


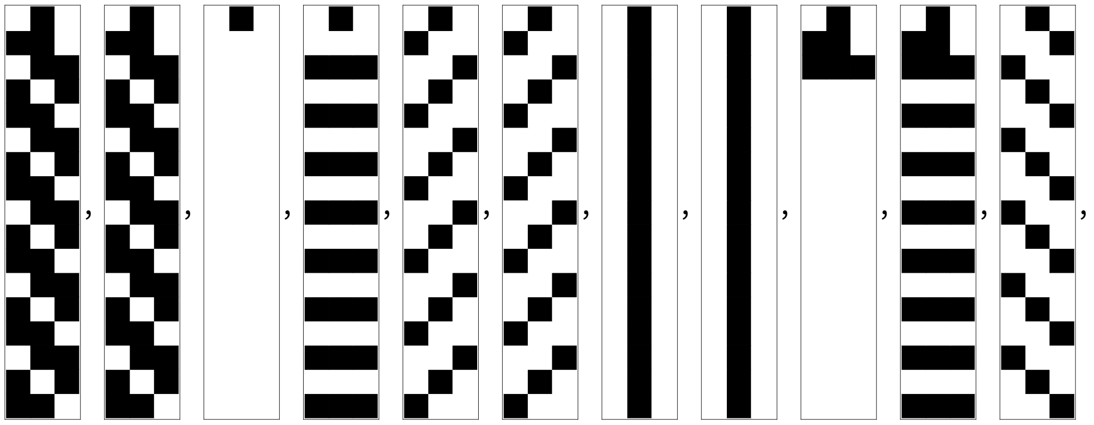


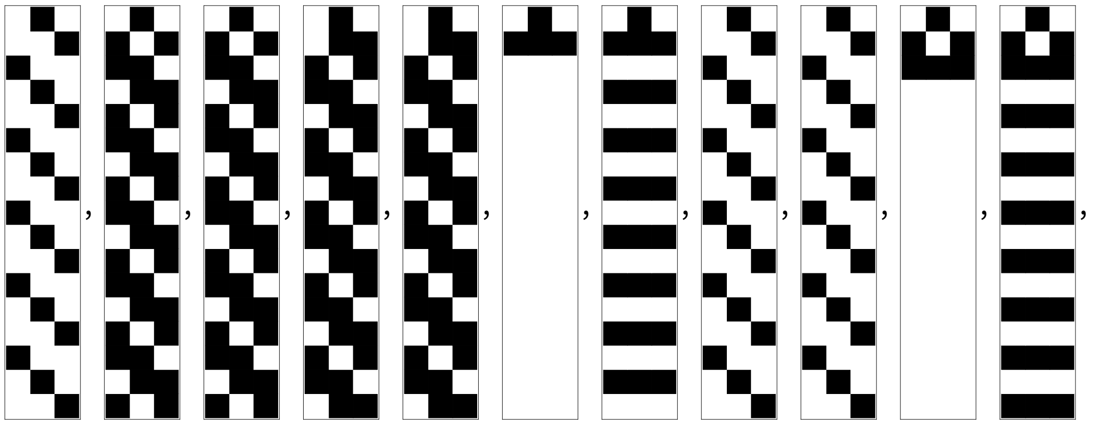


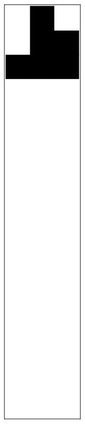
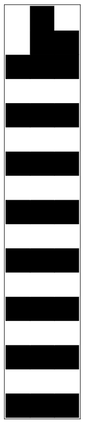
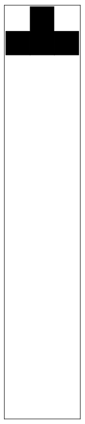
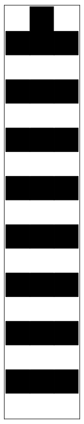
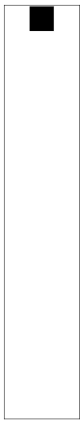
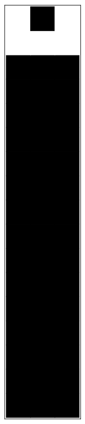
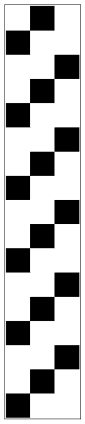
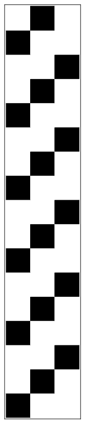
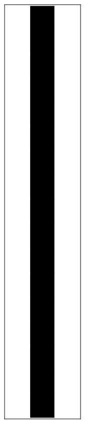
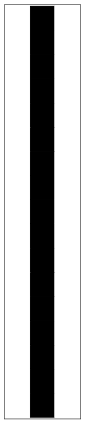
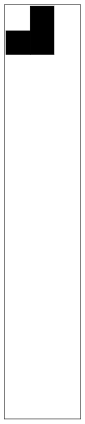


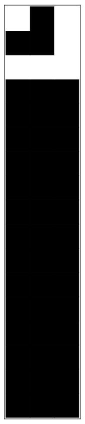
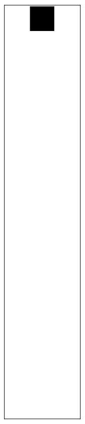
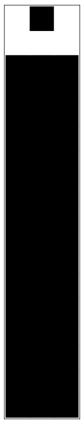
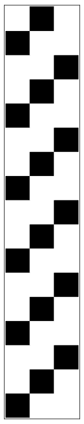
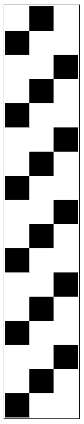
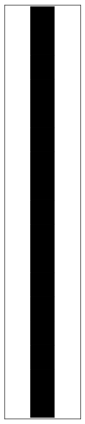
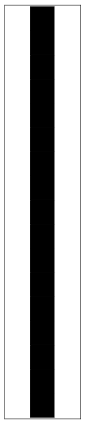
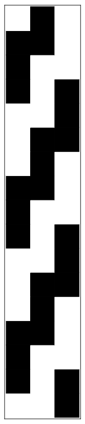
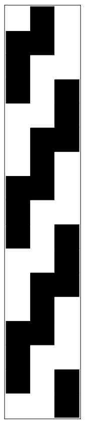
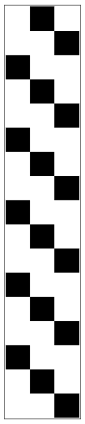
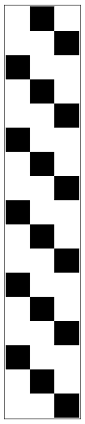


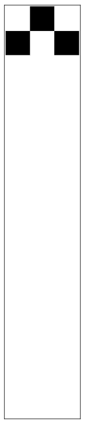
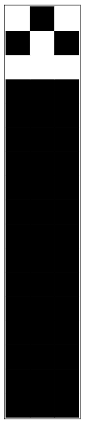
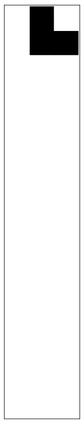
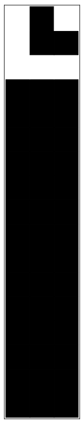
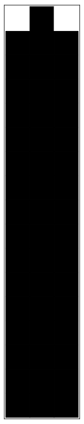
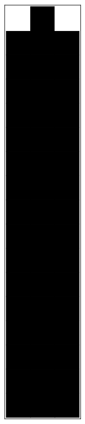
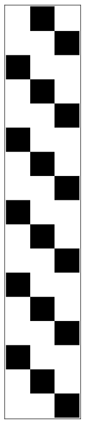
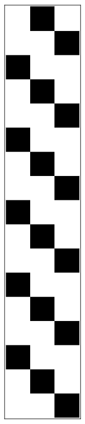
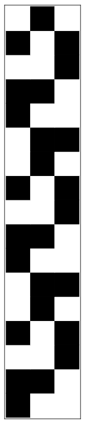
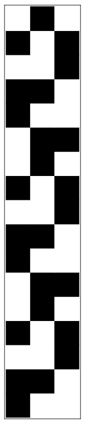
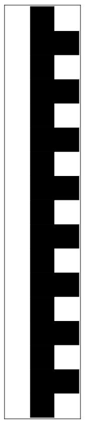


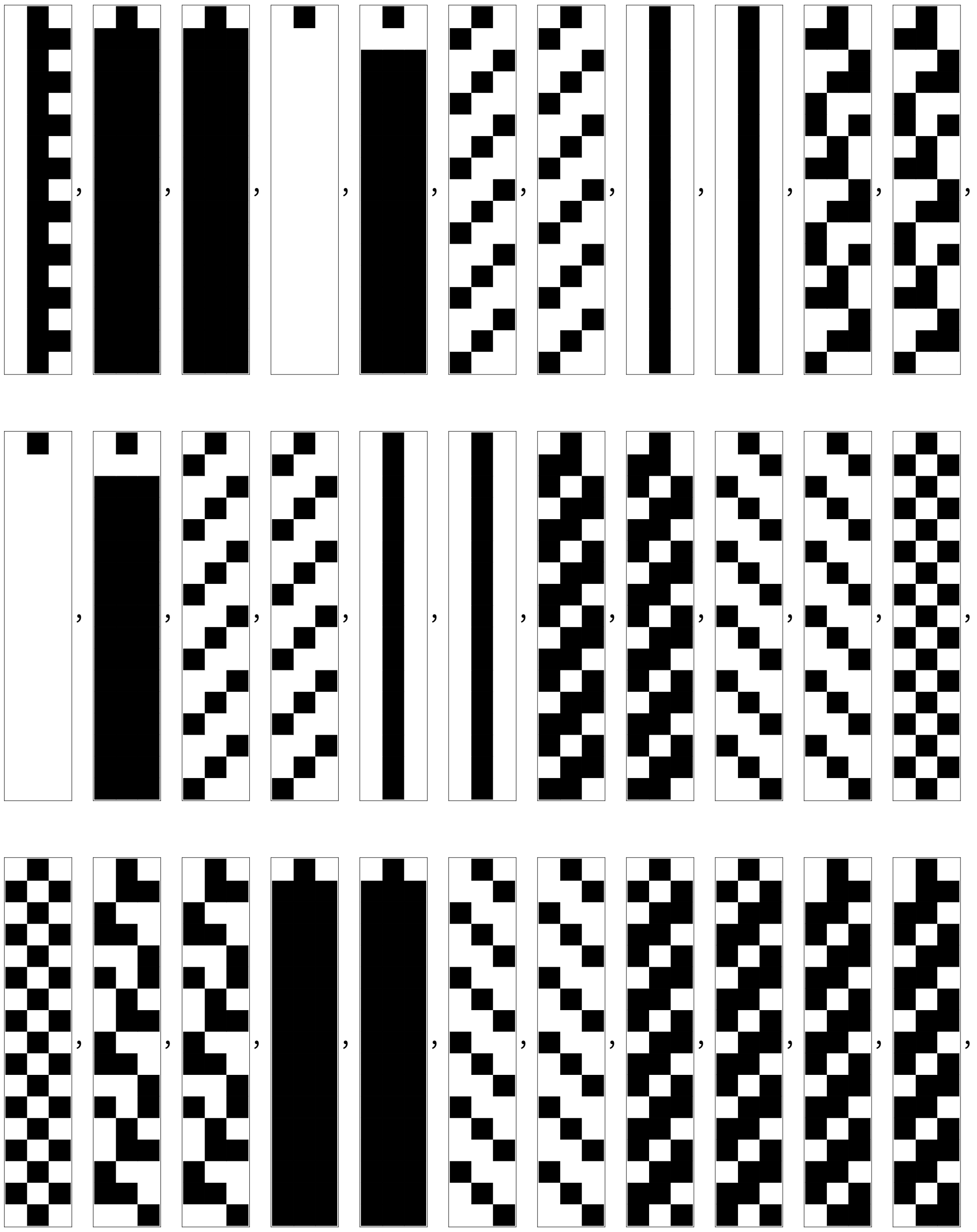


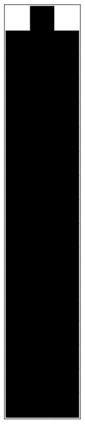
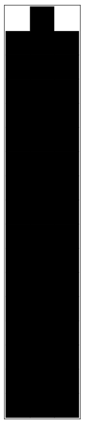
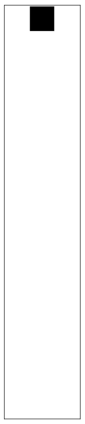
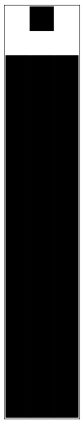
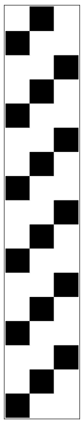
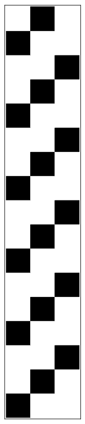
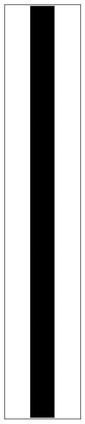
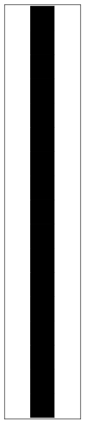
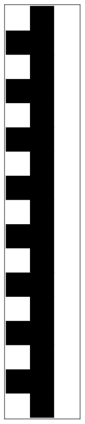
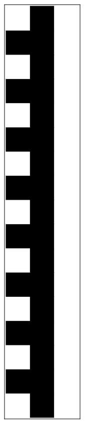
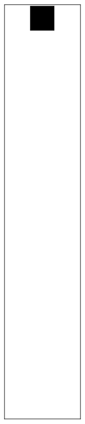


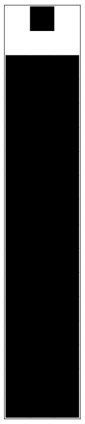
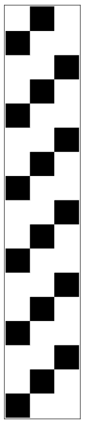
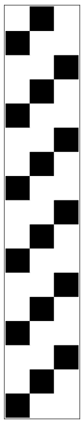
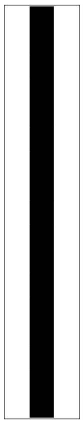
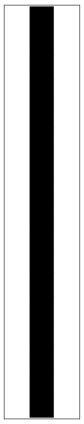
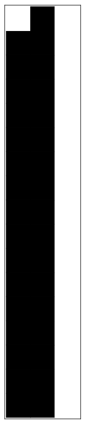
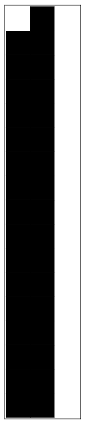
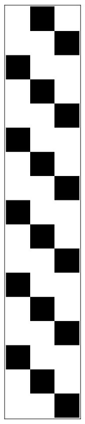
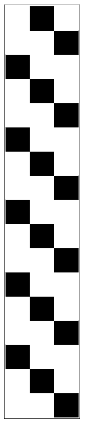
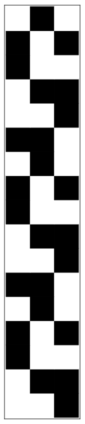
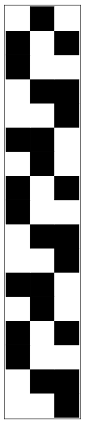
,
,
,
,
,
,
,
,
,
,
,

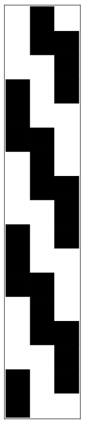
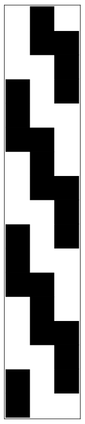
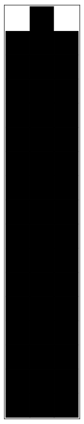
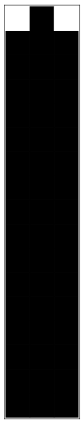
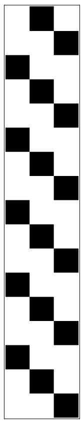
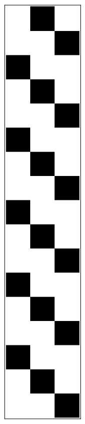
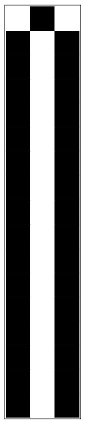
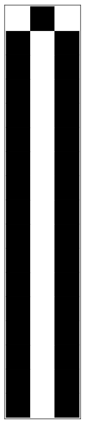
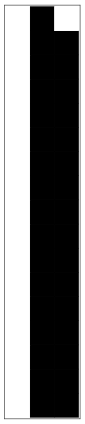
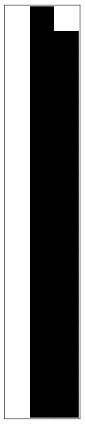
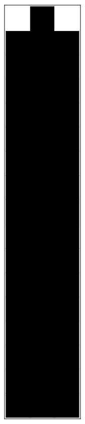
,
,
,
,
,
,
,
,
,
,
,

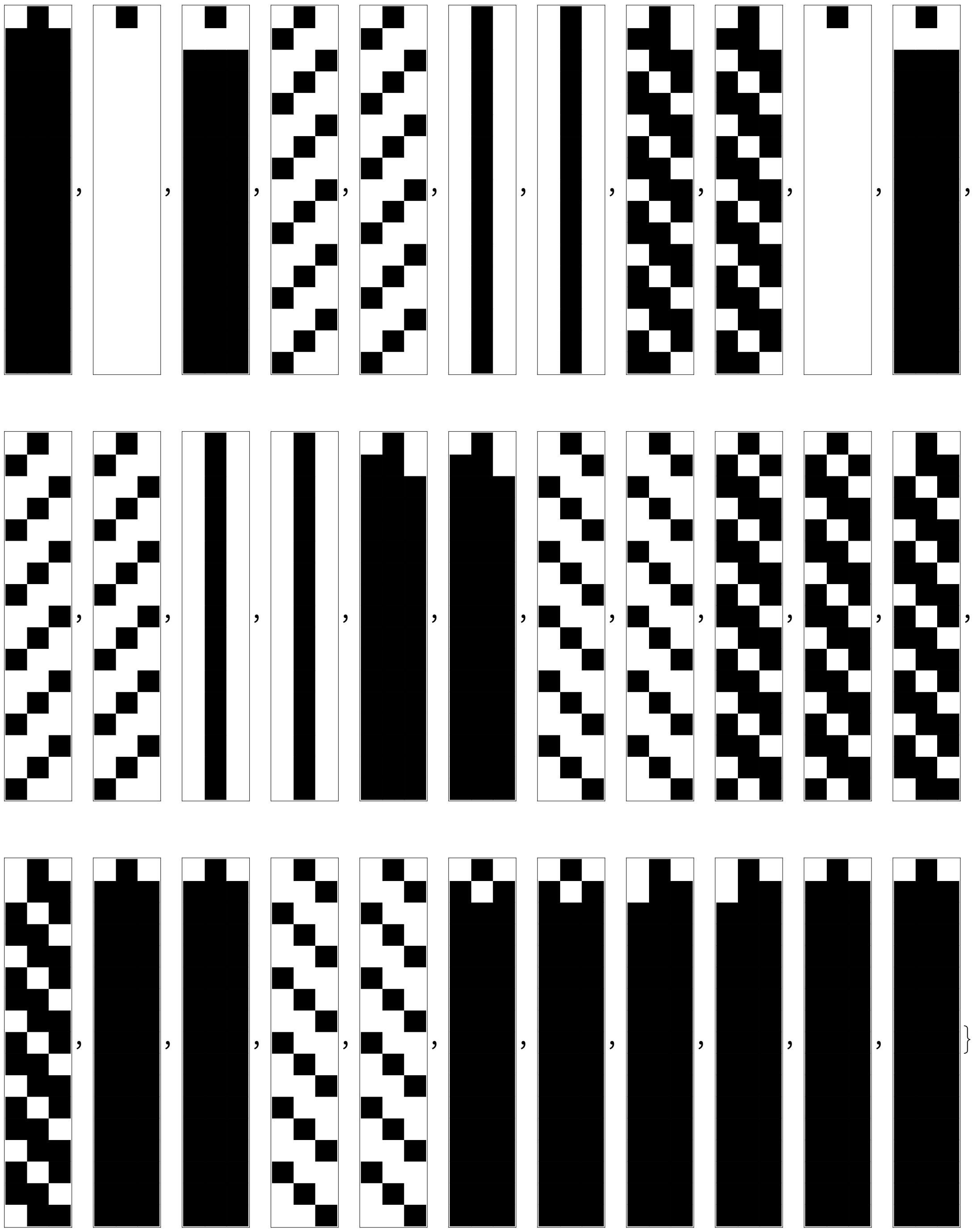
,
,
,
,
,
,
,
,
,
,
,



 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,

 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,

 ,  ,  ,  ,  ,  ,  ,  ,  ,  ,  ,



Answer 4

No, it is not possible. Think about the states of all 1’s and all 0’s. Once you reach these states, you cannot go anywhere. So it is possible to reach all possible states only once, but never over and over again. The states of all 1’s and 0’s are called *absorbing states*.