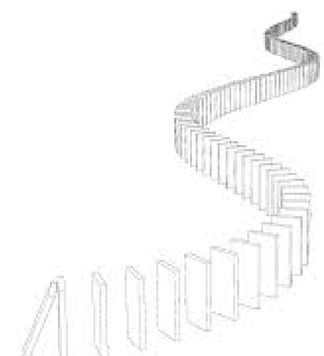
COMP 1002

Logic for Computer Scientists

Lecture 25



Admin stuff

- Assignment 4 is posted.
  - Due March 23rd.
- Monday March 20th office hours
  - From 2:30pm to 3:30pm
    - I need to attend something 2-2:30pm.

Regular expressions

- A regular expression is a standard tool for pattern matching
  - in Python, Perl, Ruby, grep, shell scripts...
  - a | b* matches either a letter a, or 0 or more repetitions of b.
  - So a regular expression defines a set of strings that it matches: a regular language.

- Recursive definition of regular expressions (as a set of strings):
  - Base: $\emptyset$, λ (empty string), all letters in alphabet
  - Recursive step: Given two regular expressions R and S, the following are regular expressions:
    - Union $R \cup S$ (sometimes written $R \mid S$)
      - Often drop parentheses when no ambiguity
    - Concatenation $R \circ S = \{xy \mid x \in R \text{ } and \text{ } y \in S\}$ (sometimes written RS)
    - A Kleene star $R^* = \{x_1 x_2 \ldots x_k \mid k \in \mathbb{N} \wedge \forall i \in \{0, \ldots, k\} \wedge x_i \in R\}$
      - k=0 ok; so zero or more strings from R concatenated together.
  - Restriction: no other strings are in the set.

Examples of regular expressions

- aa^*
  - Strings of one or more a’s.
- $(0|1)^*00$
  - Binary strings ending in 00.
- COMP(1000|1001|1002|2001)
  - Matches COMP1000, COMP1001, COMP1002 and COMP2001.
- COMP(1|2)00(0|1|2)
  - COMP1000,COMP2000,COMP1001, COMP2001, COMP1002,COMP2002
- $\emptyset$
  - Does not match anything: zero strings in the language
- λ
  - Matches the empty string: one string in the language

Permanent link to this comic: https://xkcd.com/208/

Pattern matching

- Suppose we have a DNA string:

  - AAGATTCATTAATAAATACGCTTACA
  - And a gene string ATAC
  - How do we check if the string contains the match?

AAGATTCATATAATAAATACGCTTACA
ATAC

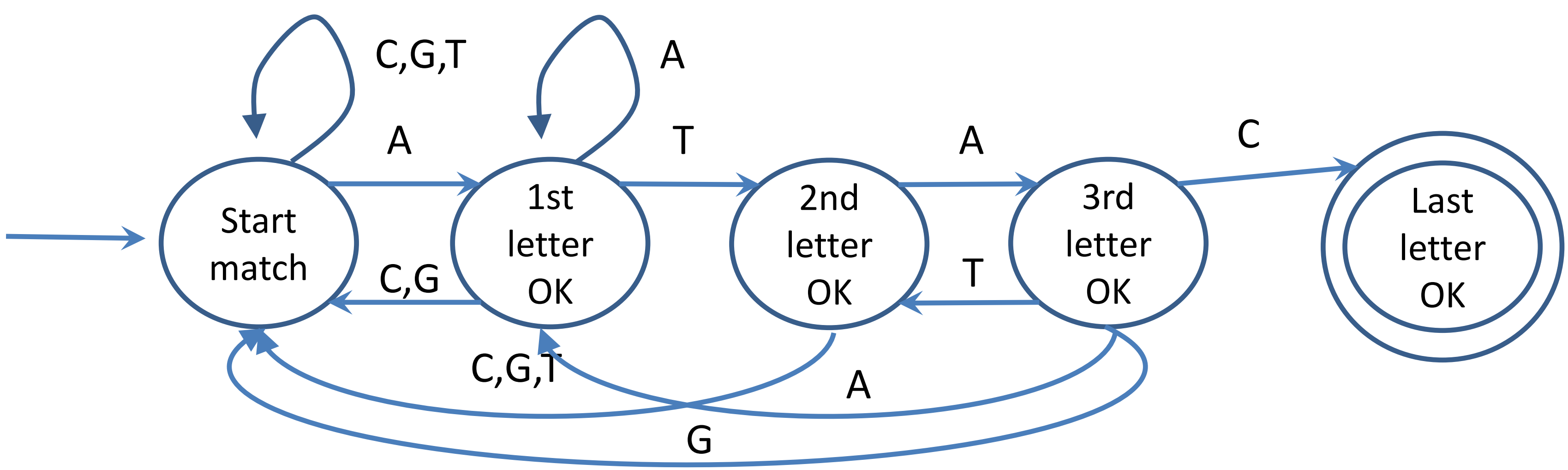
  - Could just move along checking each letter, and if mismatch, shifting by 1 character...
    - There is a faster way: finite state machines.

Matching with finite state machines

- Faster matching idea:

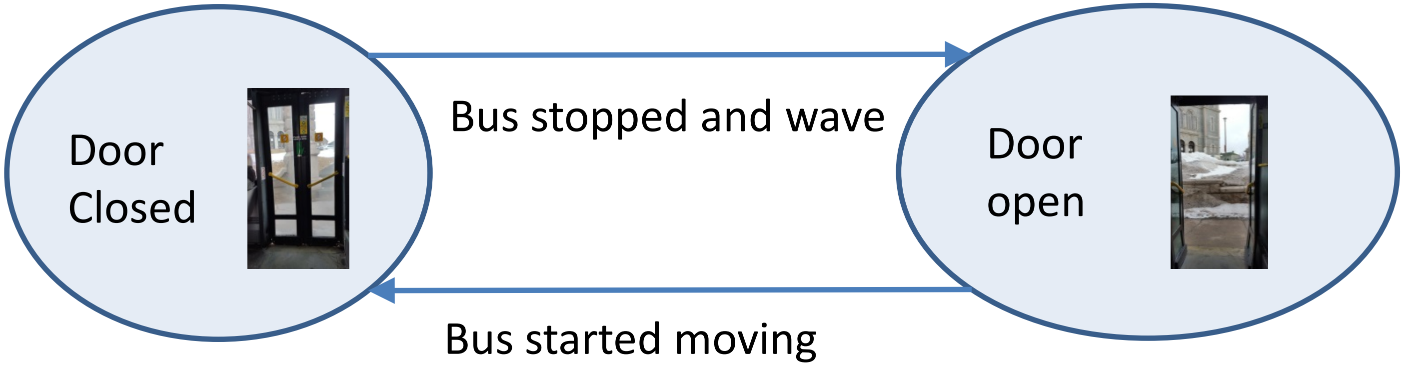
AAGATTCATATAATAAATACGCTTACA
ATAC

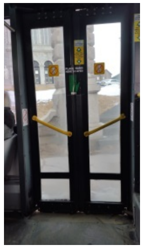
  - If mismatch T instead of C, know that shifting by 2 would be good enough; no need to re-match ATA



Finite state machine

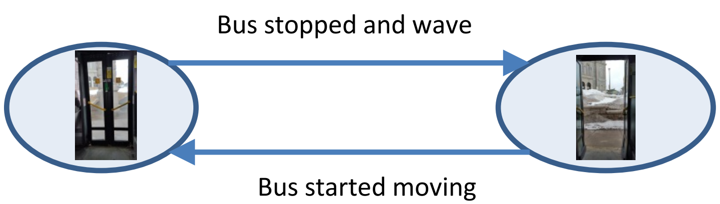
- Metrobus door: wave to open.
  - Only works when bus has stopped.
  - Description of the system:
    - If bus is in motion then closed.
    - If bus is stopped then if wave received, open.
    - If bus is stopped and there is no wave, remain closed.

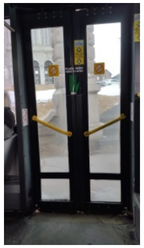




Finite state machine

- Finite state machine:
  - States
    - Including start state s, possibly finish states
  - Inputs
    - An input alphabet
  - Transitions from $States \times Inputs \rightarrow States$
    - Sometimes also have outputs:
      - Then include output alphabet
      - Transitions to $States \times Outputs$
- In the bus example
  - Two states: closed and open.
    - Looks like closed is the start state.
    - In real life, probably more states needed.
  - Input alphabet
    - Bus moving/stopping, wave.
  - Transitions:
    - If closed and stopping and sensed a wave, go to open
    - If open and started moving, go to closed.

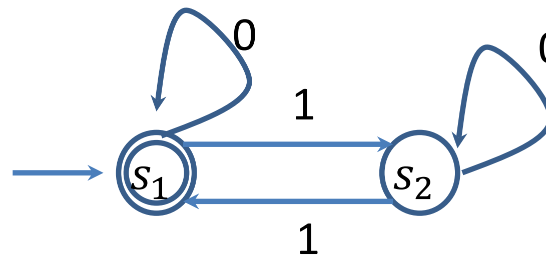




Finite automata

- Finite state machines with no output.
- Take an input string, accept if finish in an accepting state
  - Example: accept strings with even number of 1s.
    - States s_1, s_2
    - s_1 is a start state
      - Arrow
    - s_1 is an accepting state
      - Double circle
    - Input alphabet is {0,1}
    - Transitions:
      - $(s_1, 0) \rightarrow s_1$
      - $(s_1, 1) \rightarrow s_2$
      - $(s_2, 0) \rightarrow s_2$
      - $(s_2, 1) \rightarrow s_1$

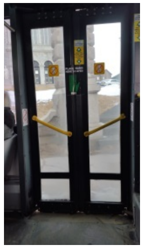
	0	1
s_1	s_1	s_2
s_2	s_2	s_1



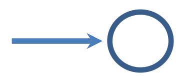
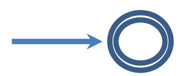
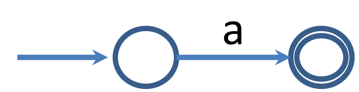
  - If exactly one transition for each pair (state, symbol)
    - Then called **deterministic finite automata (DFA)**
    - Otherwise, **non-deterministic finite automata (NFA)**
      - No transition: stop and reject. Multiple: if some choice eventually leads to accept, accept.
      - Everything an NFA can do, a DFA can do. But might need a much bigger DFA.

Regular expressions

- Recursive definition of regular expressions (as a set of strings):
  - Base: $\emptyset$, λ (empty string), all letters in alphabet
  - Recursive step: Given two regular expressions R and S, the following are regular expressions:
    - Union $R \cup S$ (sometimes written $R \mid S$)
      - Often drop parentheses when no ambiguity
    - Concatenation $R \circ S = \{xy \mid x \in R \; and \; y \in S \}$ (sometimes written RS)
    - A Kleene star $R^* = \{x_1 x_2 \ldots x_k \mid k \in \mathbb{N} \wedge \forall i \in \{0, \ldots, k\} \wedge \; x_i \in R \}$
      - k=0 ok; so zero or more strings from R concatenated together.
  - Restriction: no other strings are in the set.



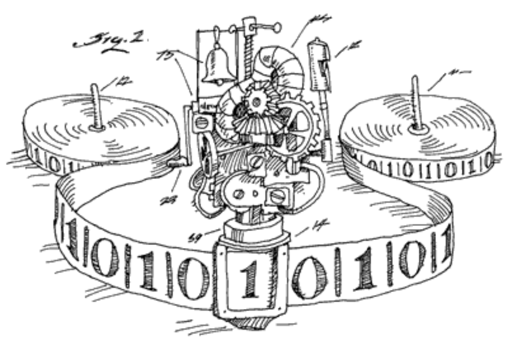
Finite automata compute what regular expressions match

- Each regular expression can be computed by a finite automaton (in particular, NFA).
- Proof (structural induction)
  - Base case:
    - Compute the empty language: 
    - Accept just the empty string: 
    - Accept just the string with one symbol a: 

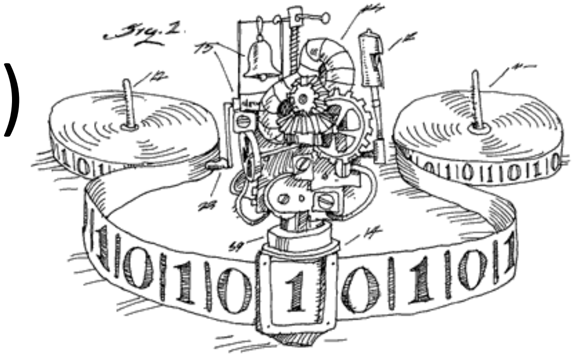
  - Recursion step: take NFAs for R and for S.
    - Kleene star R^*: loop back to start (make start accepting)
    - Union $R \cup S$: done with ambiguity (combine starts)
    - Concatenation $R \circ S$: accept states of R become start of S.

Turing machines

- Like finite automata with external memory.
- Church-Turing thesis: Turing machines can compute anything “computable”
  – In particular, anything a human can compute.

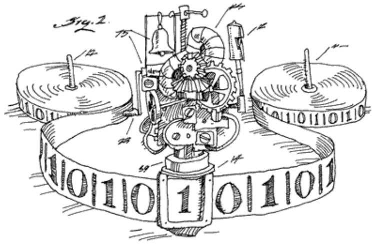


Turing machine

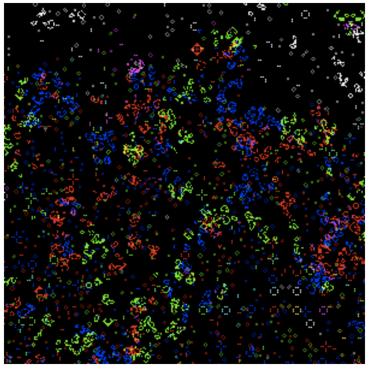


- A Turing machine has an (unlimited) memory, visualized as a tape
- Or a stack of paper
- And takes very simple instructions:
  – Read a symbol
  – Write a symbol
  – Move one step left or right on the tape
  – Change internal state.

Church-Turing thesis

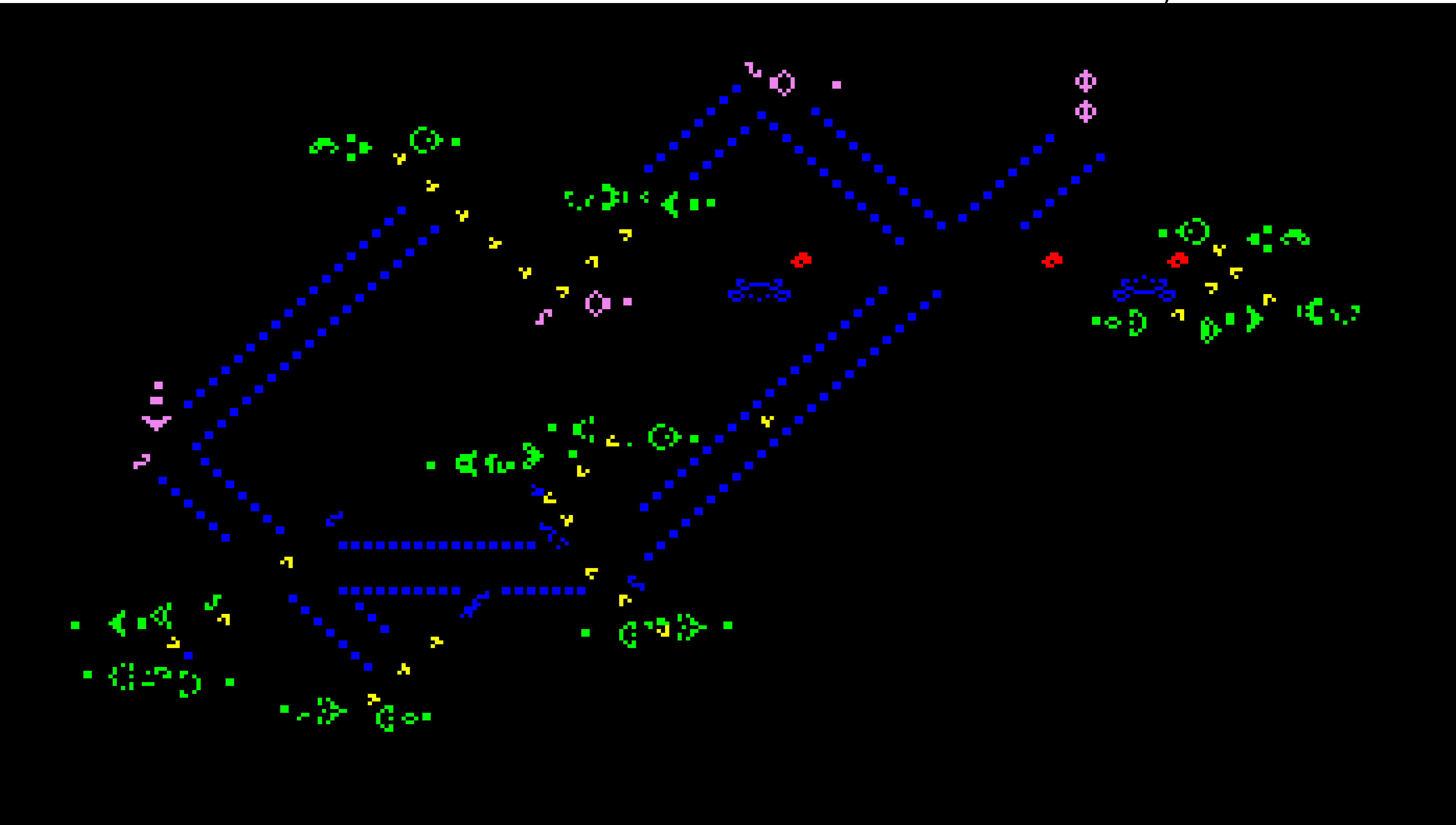


***Everything we can call "computable" is computable by a Turing machine*.**



Conway's game of life

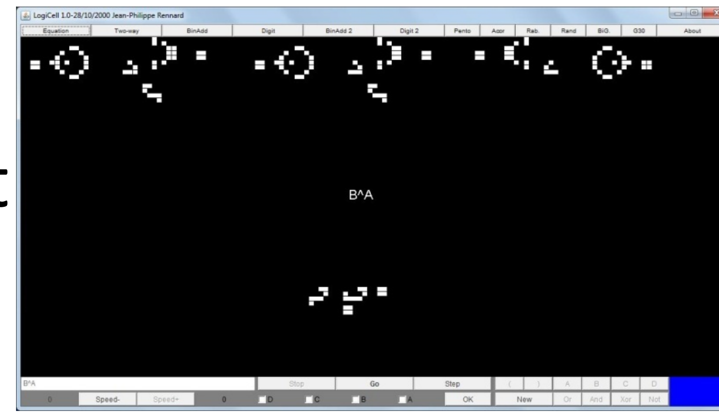
- At every step of the game:
  - Every live cell with less than 2 neighbours dies
  - Every live cell with more than 3 neighbours dies
  - A cell with exactly 3 neighbours becomes alive (is "born").

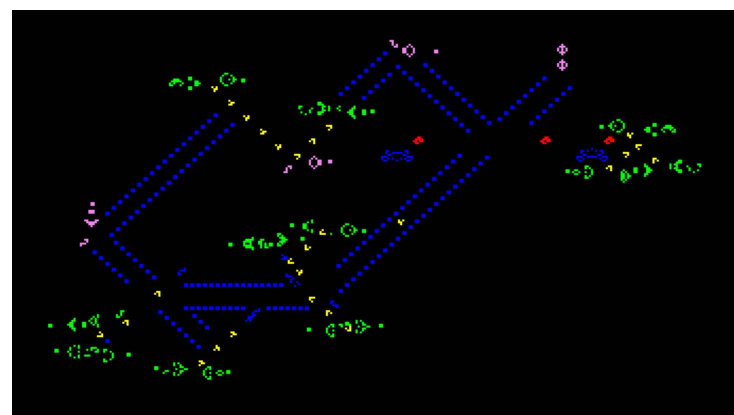


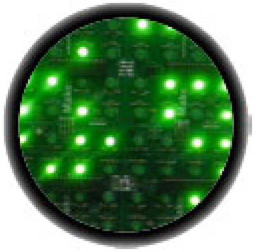
Conway's game of life: what does it mean to compute?

- Rules of the Game of Life:
- At every step of the game:
  - Every live cell with less than 2 neighbours dies
  - Every live cell with more than 3 neighbours dies
  - A cell with exactly 3 neighbours becomes alive (is "born").

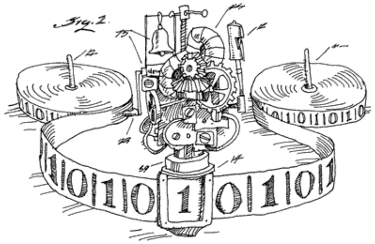
- Start with a few cells lit up
- See if cells somewhere else light up
- Make it so they only light up if some condition holds
- Just like a Turing machine going into "yes"-state if some condition holds about its input



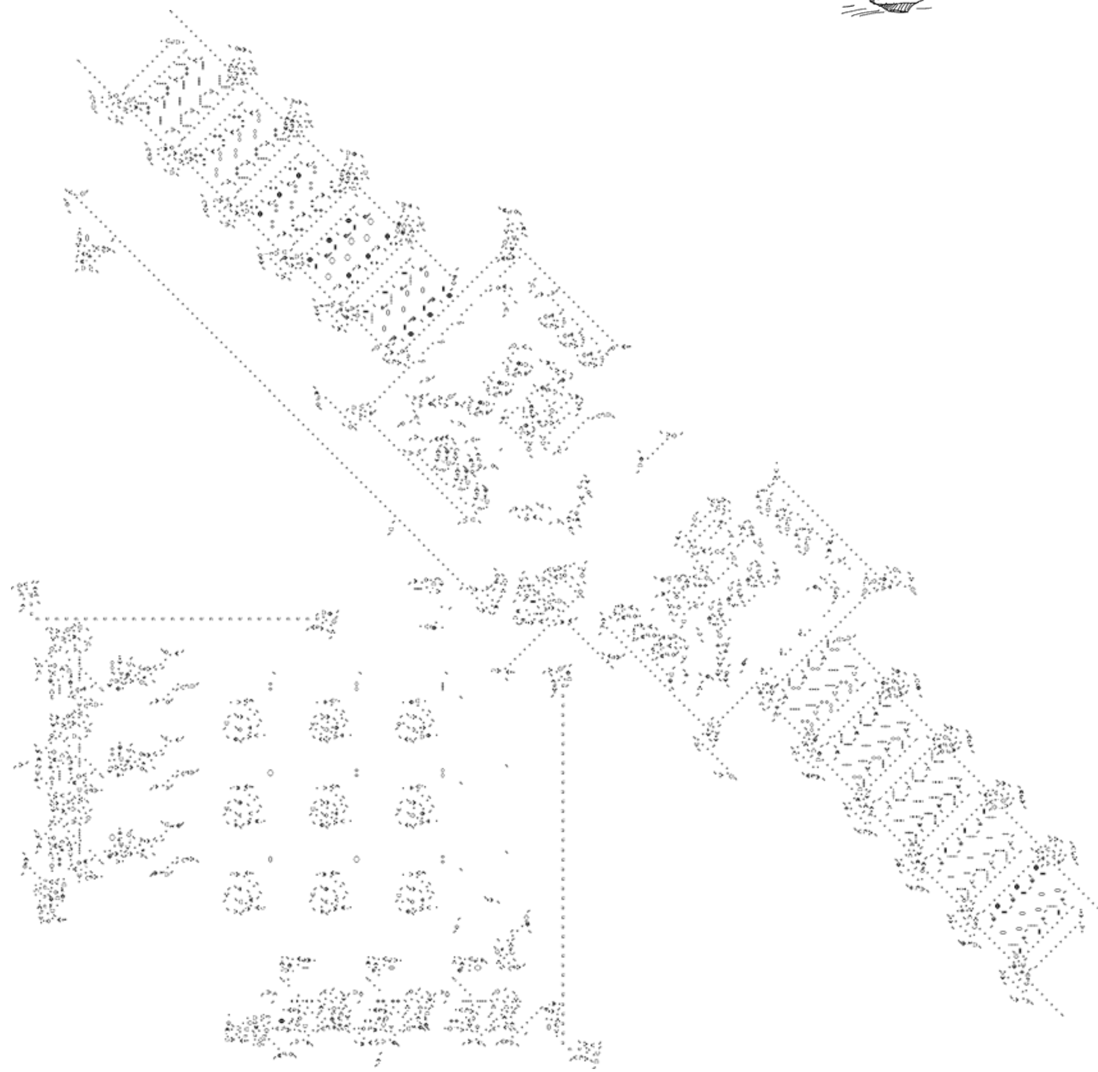




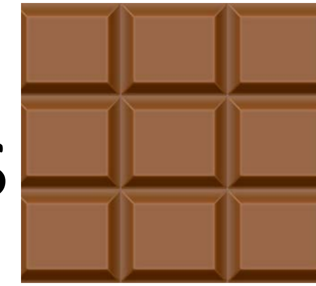
Game of life and Turing machines are equivalent



- A Turing machine can read a description of the initial configuration and keep applying the rules.
- Conway game of life can do a Turing machine using this picture:



Puzzle: chocolate squares



- Suppose you have a piece of chocolate like this:

- How many squares are in it?
  – of all sizes, from single to the whole thing