

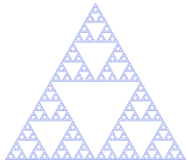


COMP 1002

Logic for Computer Scientists

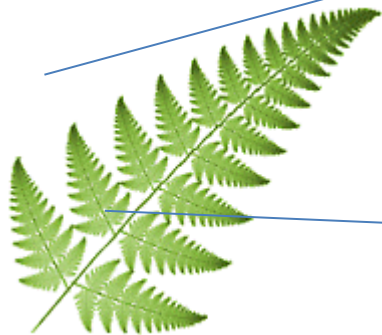
Lecture 23





Fractals

- Can use recursive definitions to define fractals
 - And draw them
 - And prove their properties.
- Fractal is a curve or geometric figure, each part of which has the same statistical character as the whole.
- Self-similar: a part looks like the whole.



Fractals in nature

- A fern leaf



- Broccoli



- Mountains



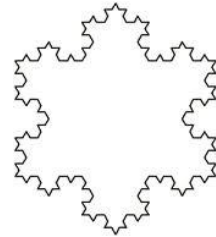
- Stock market



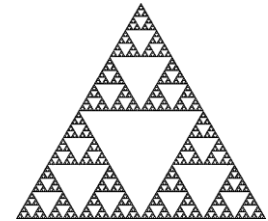
- Heart beat

Mathematical fractals

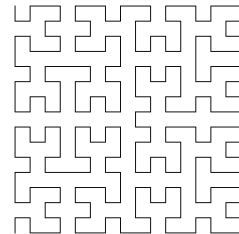
- Koch curve and snowflake



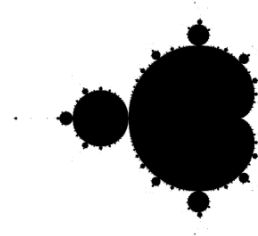
- Sierpinski triangle, pyramid, carpet



- Hilbert space-filling curve

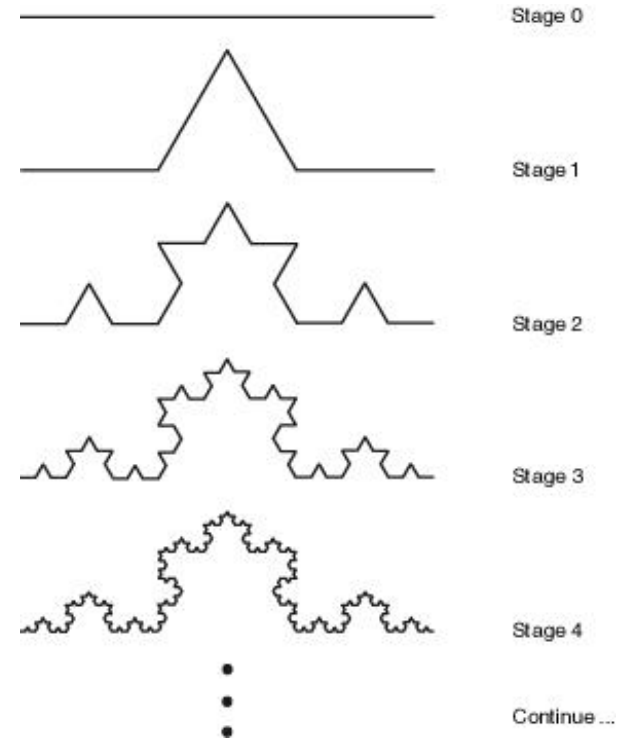


- Mandelbrot set



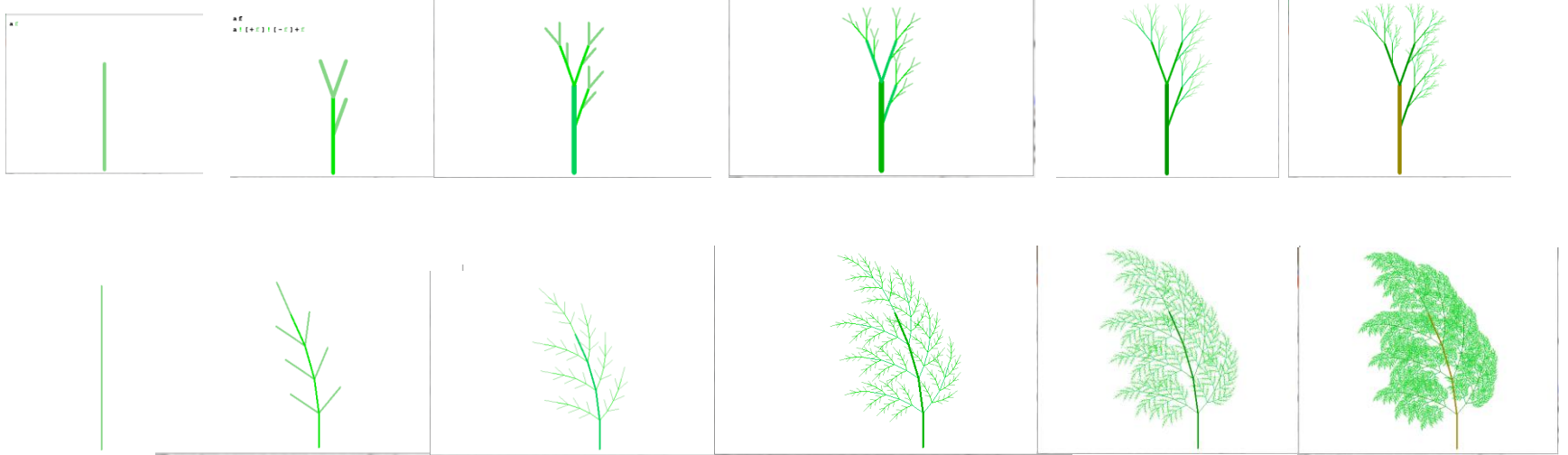
Koch curve

- *Basis:* an interval
- *Recursive step:*
Replace the inner third
of the interval with
two of the same
length
- ...



Playing with fractals

- Fractal Grower by Joel Castellanos:
- <http://www.cs.unm.edu/~joel/PaperFractal/paper.html>



Regular expressions

- A regular expression is a standard tool for pattern matching
 - in Python, Ruby, grep, shell scripts...
 - $a \mid b^*$ 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 $R \circ S = \{xy \mid x \in R \text{ and } y \in S\}$ (sometimes written RS)
 - A Kleene star $R^* = \{x_1x_2 \dots x_k \mid k \in \mathbb{N} \wedge \forall i \in \{0, \dots, k\} \wedge x_i \in R\}$
 - 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.
- $\text{COMP}(1000|1001|1002|2001)$
 - Matches COMP1000, COMP1001, COMP1002 and COMP2001.
- $\text{COMP}(1|2)00(0|1|2)$
 - 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

WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.

OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!



BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!

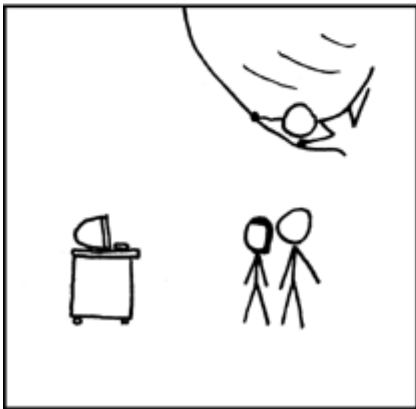

IT'S HOPELESS!



EVERYBODY STAND BACK.

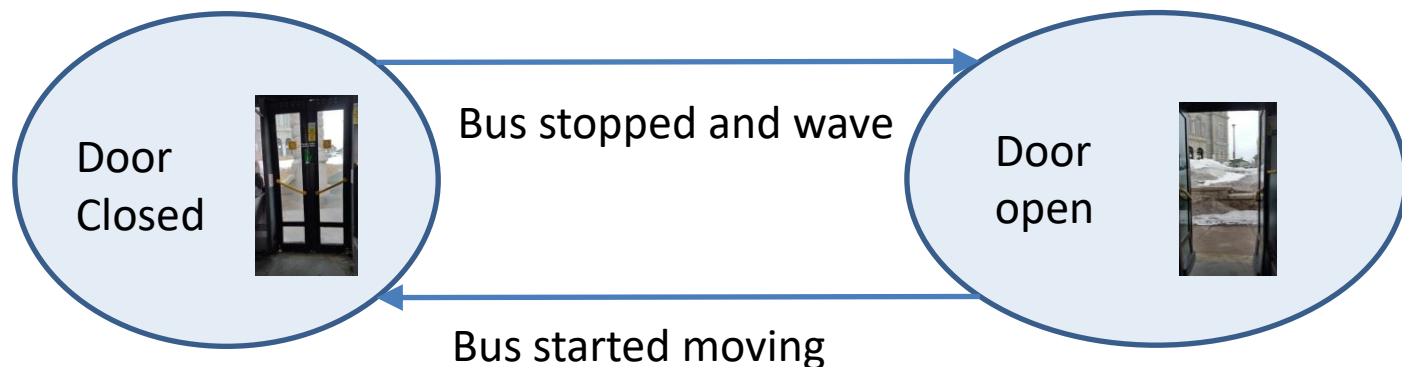
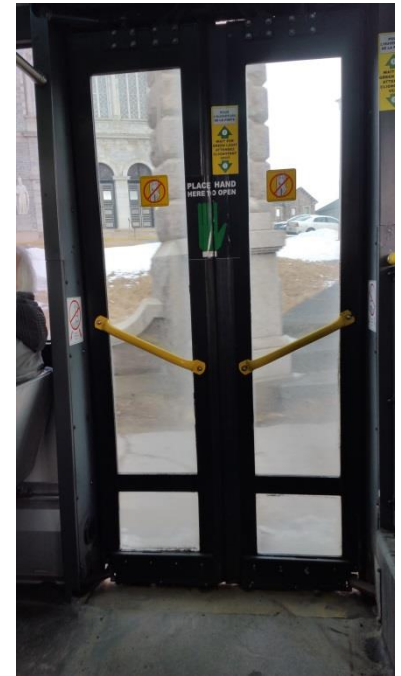


I KNOW REGULAR EXPRESSIONS.



Finite state machine

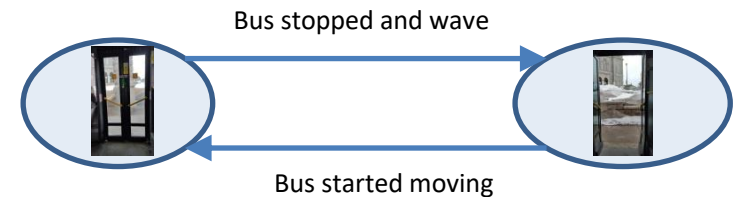
- Metrobus door: wave to open.
 - Only works when bus has stopped.
 - Description of the system:
 - If bus is in motion then closed.
 - If bus is stopped then if wave received, open.
 - If bus is stopped and there is no wave, remain closed.






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.

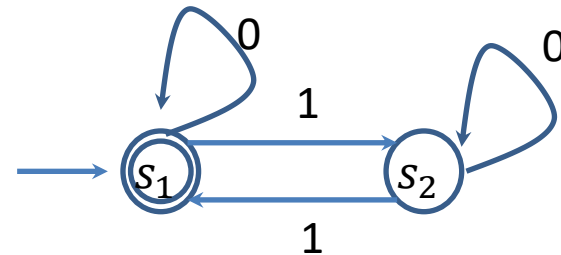
- States s_1, s_2
- s_1 is a start state
 - Arrow
- s_1 is an accepting state
 - Double circle
- Input alphabet is $\{0,1\}$



- 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 $R \circ S = \{xy \mid x \in R \text{ and } y \in S\}$ (sometimes written RS)
 - A Kleene star $R^* = \{x_1 x_2 \dots x_k \mid k \in \mathbb{N} \wedge \forall i \in \{0, \dots, k\} \wedge x_i \in R\}$
 - $k=0$ 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 .

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

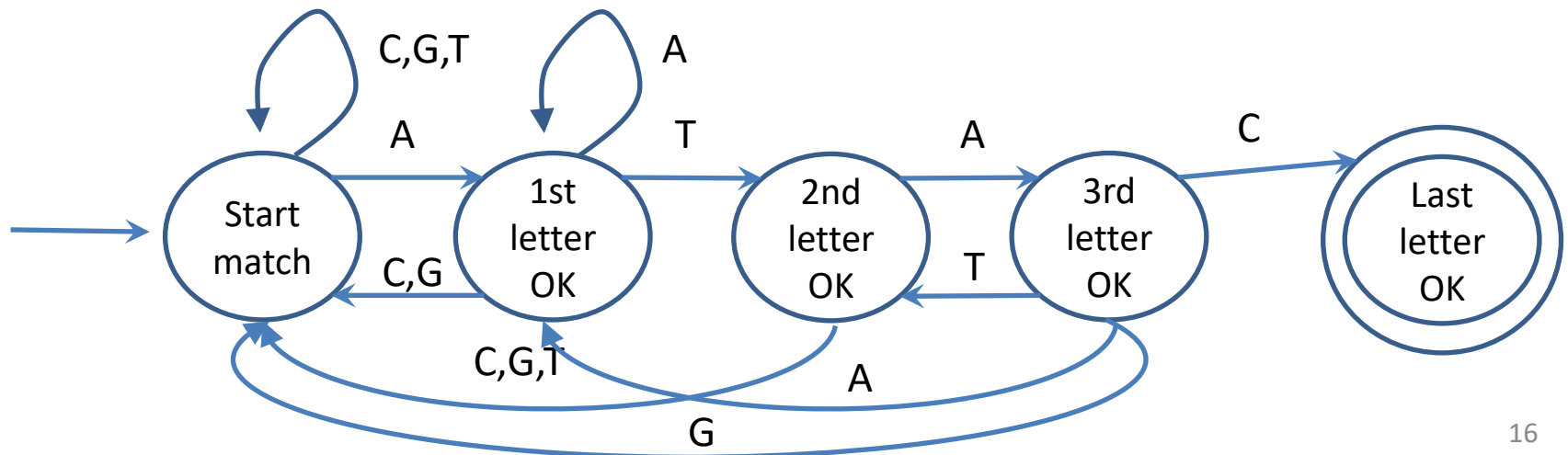
- Could just move along checking each letter, and if mismatch, shifting by 1 character...
 - There is a faster way: finite state machines.

Matching with finite state machines

- Faster matching idea:

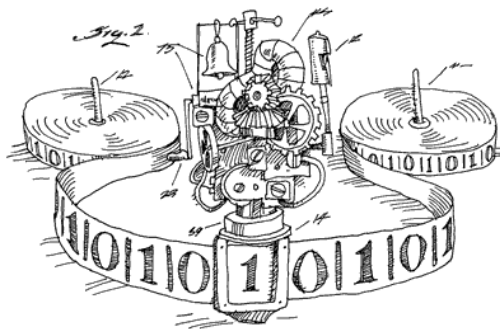
AAGATTCATATAATAAATACGCTTACA
ATAC

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



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

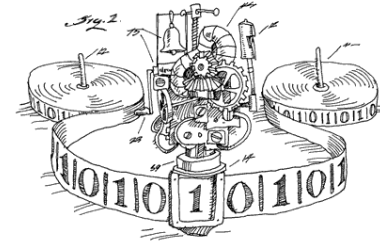
- is a mathematical model of computation that defines an abstract machine which manipulates symbols on a strip of tape according to a table of rules.
- Despite the model's simplicity, given any computer algorithm, a Turing machine can be constructed that is capable of simulating that algorithm's logic.

A Turing machine consists of:

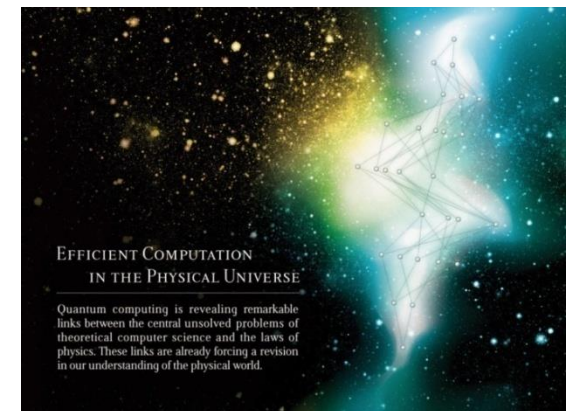
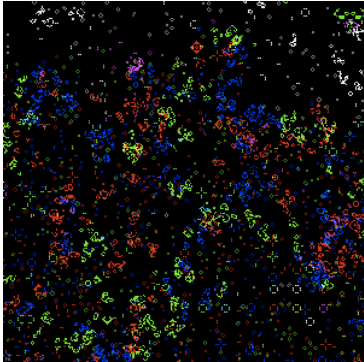
- A **tape** divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special *blank* symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right. Cells that have not been written before are assumed to be filled with the blank symbol.
- A **head** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.
- A **state register** that stores the state of the Turing machine, one of finitely many. Among these is the special *start state* with which the state register is initialized.
- A finite **table** of instructions that, given the *state*(q_i) the machine is currently in *and* the *symbol*(a_j) it is reading on the tape (symbol currently under the head), tells the machine to do the following *in sequence*:
 - Either erase or write a symbol (replacing a_j with a_{j1}).
 - Move the head (which is described by d_k and can have values: 'L' for one step left *or* 'R' for one step right *or* 'N' for staying in the same place).
 - Assume the same *or* a *new state* as prescribed (go to state q_{i1}).



Church-Turing thesis



Everything we can call “computable” is computable by a Turing machine.





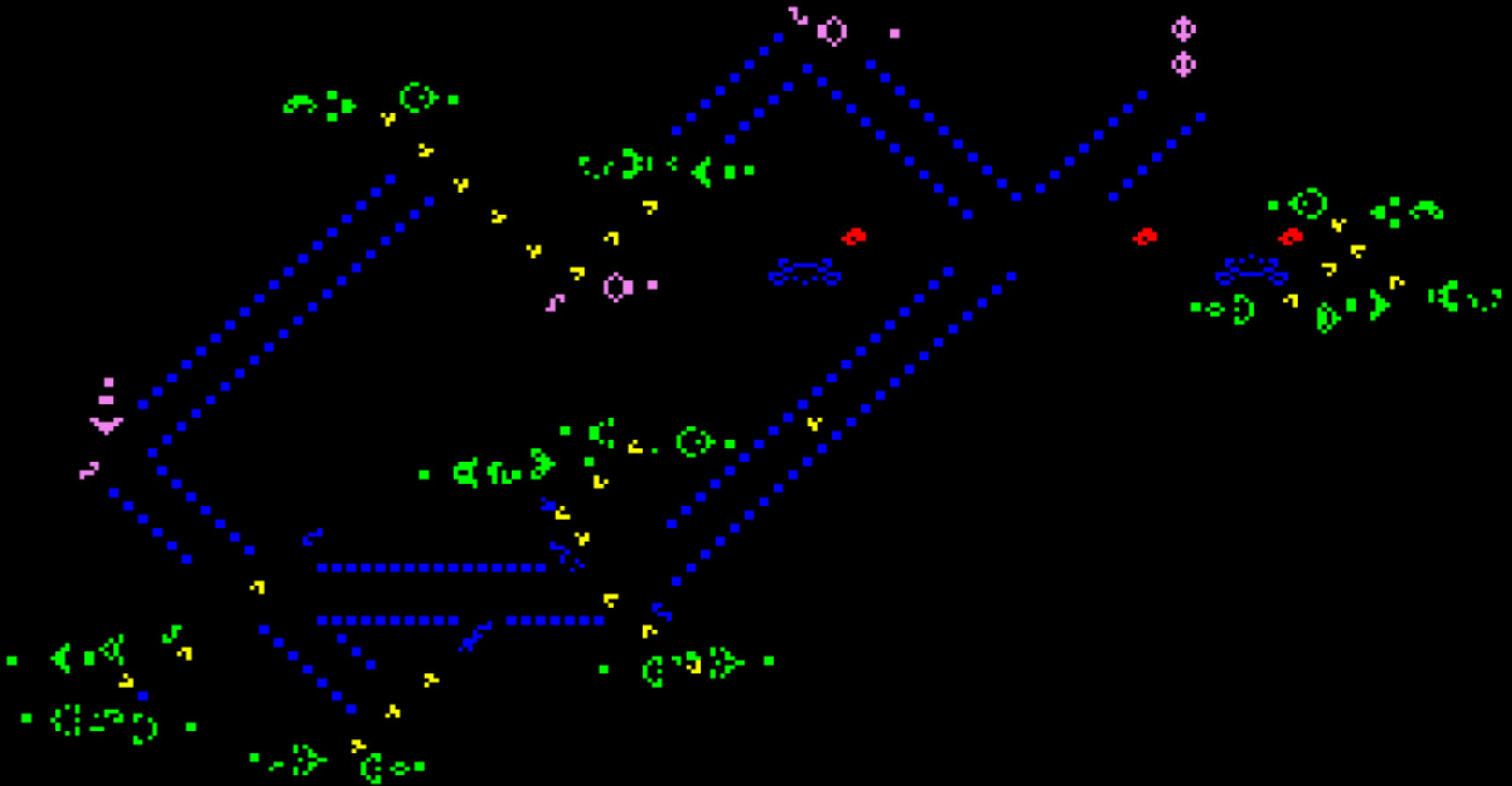
Puzzle: coins



- A not-too-far-away country recently got rid of a penny coin, and now everything needs to be rounded to the nearest multiple of 5 cents...
 - Suppose that instead of just dropping the penny, they would introduce a 3 cent coin.
 - Like British three pence.
 - What is the largest amount that cannot be paid by using only existing coins (5, 10, 25) and a 3c coin?

Conway's game of life

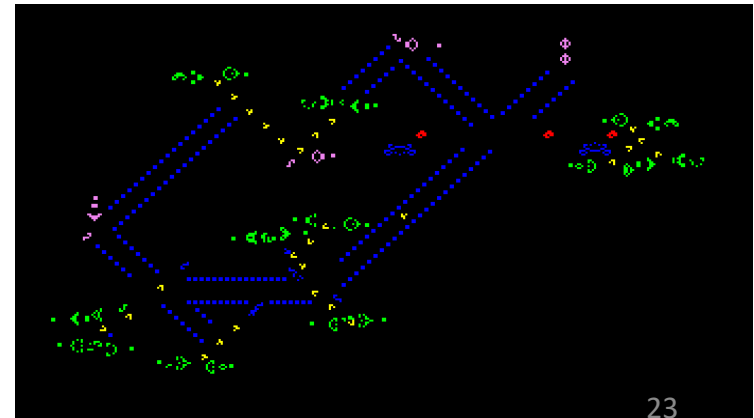
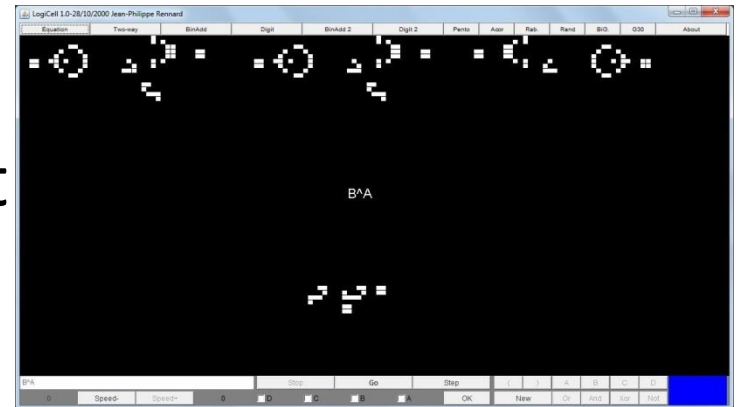
- At every step of the game:
 - Every live cell with less than 2 neighbours dies
 - Every live cell with more than 3 neighbours dies
 - A cell with exactly 3 neighbours becomes alive (is "born").



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

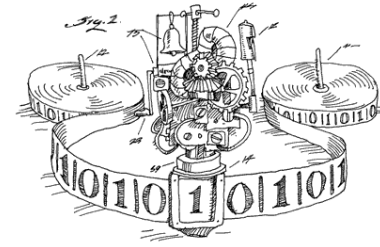
- Rules of the Game of Life:
- At every step of the game:
 - Every live cell with less than 2 neighbours dies
 - Every live cell with more than 3 neighbours dies
 - A cell with exactly 3 neighbours becomes alive (is “born”).

- 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:

