Turing machine
- a finite alphabet
- a finite set of states
- a single, current state
- a single pointer to the current cell on the tape
- a table specifying an action given a (current) state and the (current) value under the tape
- designated "blank" symbol in the alphabet, which is the only symbol which may appear infinitely many times as input to the machine

- has an infinite tape of discrete cells, each of which contains a single symbol from a finite alphabet

An action involves writing a symbol on the current cell, transitioning to a new state, and either:
- moving the tape left
- moving the tape right
- halting.

Turing machine for DFA
- States will be the same
- Alphabet: same + a special "blank" symbol
- Input: ... blank blank blank input string blank blank ...

For each arrow in our DFA diagram:
- action[state, symbol] = write "blank", goto next state (of the arrow), advance the input

For each final state in our DFA:
- action[state, blank] = write any non-blank symbol, stay on the same state, halt

All other actions/entries in the table = write blank, stay on the same state, halt

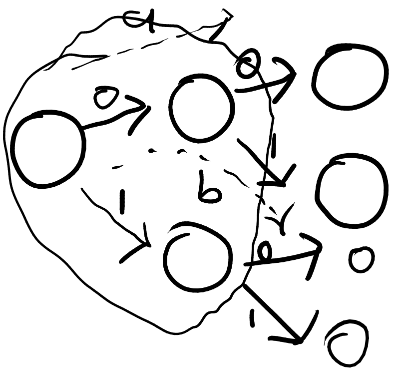
Have a turing machine with alphabet Sigma, |Sigma| > 2.
We want to make a conceptually equivalent turing machine that uses an alphabet of { 0, 1 }.

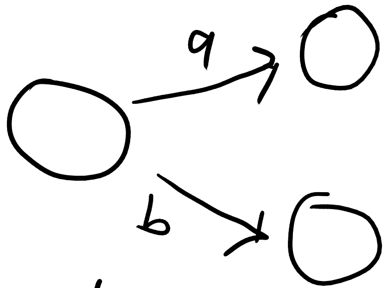
n = log_2(|Sigma|) // round up

n bits can represent 2^n >= |Sigma| distinct things.

Sigma = { a, b, c, d }

00 01 10 11





actions for c, d exist too (not shown)

Every turing machine has an equivalent turing machine that uses an alphabet of { 0, 1 }.
Every turing machine (reduced to an alphabet of { 0, 1 }) can be represented uniquely as a sequence of bits

Suppose there exists a turing machine with the following properties ("HALT"):
- conceptually it takes two inputs (on a single tape)
- the first input will be interpreted as a description of a turing machine
- the second input will be the input to the described turing machine
- always halt (in finite steps)
- halt with 1 if the first input is a valid description of a turing machine,
  and the described TM would halt given what was the second input
- halt with 0 otherwise

Then HALT' exists, that functionally computes the following.
HALT'(x) = HALT(x, x)

1. What we want is to turn the input tape conceptually consisting of a single sequence of bits into an input tape consisting of a pair of sequence of bits.
2. Then we "just have" the original HALT TM.

Then a 3rd TM exists as follows:
- Copy the table of HALT'.
- Create a new state (row) "spin".
- "spin" will always transition to itself.
- Any entry in the original HALT' that was "halt with 1", transition to "spin".
- Any entry in the original HALT' that was "halt with 0" can be left as is.

Call this machine "SPOILER".

What happens when we give (SPOILER, SPOILER) as an input to HALT?
- If HALT returns 1, then that means SPOILER(SPOILER) should halt.
    - HALT'(SPOILER) = 1
    - SPOILER(SPOILER) would end up not halting.
    - Contradiction.
- If HALT returns 0, then that means SPOILER(SPOILER) doesn't halt.
    - HALT'(SPOILER) = 0
    - SPOILER(SPOILER) would also halt with 0.
    - Contradiction.

No such TM "HALT" can exist.

Matiyasevich and O'Rear 2016: 744 state TM that halts iff the riemann hypothesis is false.

"Busy Beaver" function is defined as follows.
BB(n, m) := the largest number of non-blank characters a TM with n symbols and m states can produce
(after halting). Initially the tape is all blank.

BB(2, 1) = 0
BB(2, 2) = 4
BB(2, 3) = 6
BB(2, 4) = 13

For n = 2 (symbols): for m > 4 the BB(n, m) is not known.
The current lower bound for BB(2, 6) is 10^{18627}.

BB(2, 744) is unimaginably large.

- jump(offset)
- branch(state, symbol, offset)
- left(symbol, next_state)
- right(symbol, next_state)
- halt(symbol)