Computability Theory

- We'll finish the class w/ a bang: theory of computability.

- motivating Q: can we mathematically formalize notion of an algorithm
  (aka "computer program"
  aka "mechanical procedure")

- follow up Q: which questions can be solved by an algorithm? which can't?

- e.g. is there an algorithm that...
  ... decides whether a given prop'l fmla A is satisfiable?
  (yes! "compute the truth table")
  ... decides whether a given multivar polynomial $p(x_1,\ldots,x_n) \in \mathbb{Z}[x_1,\ldots,x_n]$ has an integer soln? (integer coefficients)
  (Hilbert's 10th — turns out answer is no!)
  ... decides whether a given first-order fmla A is satisfiable?
  (... TBD!)

- 2nd follow up: for those problems that can be solved algorithmically, which can be solved quickly?

Decision problems

Def'n an *alphabet* is a nonempty set A, whose elements we call *symbols*

↳ will usually consider *finite* alphabets
$A = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$

Def'n Sps A is an alphabet.
- a *word* (or *string*) from A is a finite sequence $a_1 a_2 \dots a_n$ of symbols $a_i \in A$
- if $w = a_1 \dots a_n$ is a word we write $n = |w|$ for the *length* of w.
- write A^n for the set of all words of length n from A
  ↳ *note*: unique word of length 0 is the empty word ϕ.
  ↳ Thus $A^0 = \{\phi\}$
- define $A^* = \bigcup_n A^n_{\bullet} =$ set of all finite words from A

Ex (1) if $A = \{0,1\}$ then
$A^0 = \{\phi\}$, $A^1 = \{0,1\}$, $A^2 = \{00, 01, 10, 11\}$
...
$A^* = \bigcup A^n_{\bullet} = \{\phi, 0, 1, 00, 01, 10, 11, 000, 001, \dots\}$

② if $A = \{0, 1, \ldots, 9\}$
then $A^* = \{\phi, 0, 1, \ldots, 9, 10, 11, \ldots, 20, 21, \ldots\}$

Def'n A decision problem (A, P) consists of a finite alphabet A and a set of words $P \subseteq A^*$

↳ the "problem" for a decision problem (A, P) is: is there an algorithm that decides when a given $w \in A^*$ belongs to P? (... more later)

Ex Sps $L = \{F_1, \ldots, F_n, R_1, \ldots, R_m\}$ is a finite language.
- if we represent every variable x_n by the string xn w/ n written in binary we can view every WFF in L as a word in the finite alphabet $A = L \cup \{x, 0, 1, \neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \exists, \forall, (,), =\}$
- We VALIDITY$_L$ $= (A, P)$ denote the decision problem of determining which L-formulas are logically valid
i.e. $P = \{S : S$ is an L-sentence and $\models S\}$

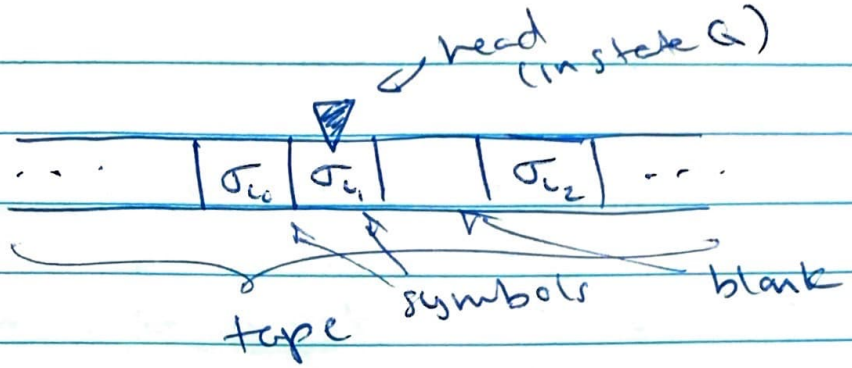
Turing machines:

- Turing machines (defined below) are a primitive formal model of a computer (i.e. a device that can execute algorithms)
- but turns out: in terms of "what algorithms they can run" TMs can do as much as possible!

- intuitive description: a TM consists of:
  - a bi-infinite tape consisting of squares in each of which is a single symbol (or nothing)
  - a read/write head that at a given moment is over a square and is in a given state
  - a set of instructions (the program) of the form

    "if in state (blah) and current square reads (bleh), overwrite (bluh), move (left/right/stay), and go to state (blof).

    ... and STOP if we reach a halting state"



more formally:

Def'n A **Turing machine** (TM) on an alphabet $A = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ consists of: (i) a finite set of **states** $\{Q_0, Q_1, \ldots, Q_m\}$

Q_0 is the **start state**

(ii) a **table** of $(m+1)(k+1)$-many 5-tuples called **instructions** of the form $(Q_i, a, b, s, Q_j]$ where:

↳ $a, b \in A \cup \{*\}$ where $*$ is a formal symbol denoting a blank.

(sometimes let $\sigma_0 = *$ so that $A \cup \{*\} = \{\sigma_0, \sigma_1, \ldots, \sigma_k\}$)

Q_m is the **halting state**

↳ for each $0 \le i \le m$ and $\sigma_j \in A \cup \{*\}$ there is exactly one tuple of the form $(Q_i, \sigma_j, b, s, Q_j)$

↳ $s \in \{-1, 0, 1\}$

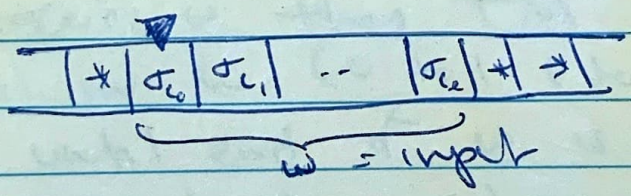
- intuitively the instruction (Q_i, a, b, s, Q_j) says:

"if in state Q_i and current square reads a, overwrite b, move s, and go to state Q_j"

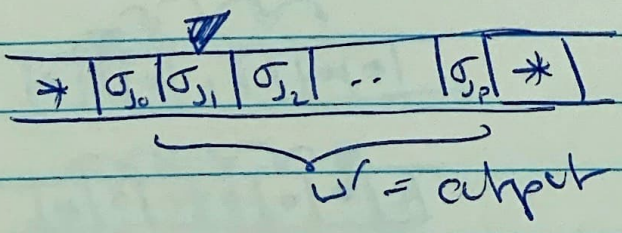
where $s = -1/0/1$ means left/stay/right respectively

- given an input word $w \in A^*$, the computation of a TM proceeds like:
  ① write the symbols in $w = \sigma_{i_0}\sigma_{i_1}\dots\sigma_{i_e}$ on consecutive squares of the tape (other squares empty, i.e. have a *)
  ② put r/w head over σ_{i_0} square (beginning of word) in state q_0 and run program until the step state q_m is reached (or indefinitely, if step state is never reached)

start:



③ the output is the word w' that begins in the square over which the r/w head finishes (reading right)



Ex Consider the TM on the alphabet $A = \{0,1\}$

States: Q_0, Q_1, Q_2

table

$(Q_0, a, 1-a, +1, Q_0)$ for all $a \in \{0,1\} = A$

$(Q_0, *, *, -1, Q_1)$

$(Q_1, b, b, -1, Q_1)$ for $b \in \{0,1\}$

$(Q_1, *, *, +1, Q_2)$

(Q_2, a, a, a, Q_2) $a \in A \cup \{*\}$

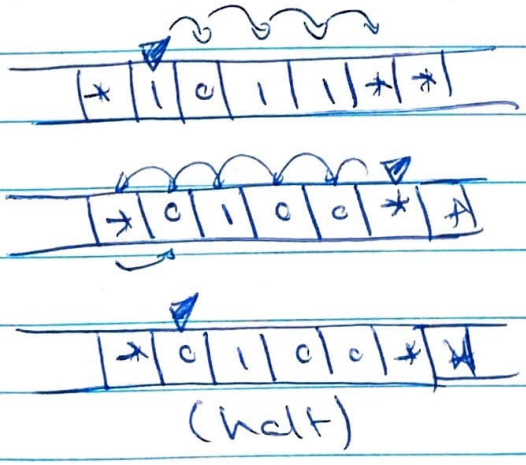
flip each symbol until end of word

turn around, go back to beginning and halt

HALT!

this part of the instruction irrelevant

- on input w this TM flips each symbol in w (0 to 1, 1 to 0) proceeding right until end of w is reached
- then scans back to beginning (w/o changing symbols) and halts
- e.g. on input $w = 1011$
  output is $w' = 0100$



Ex Now consider $A = \{1\}$ and following TM: states: Q_0, Q_1

table: $(Q_0, 1, 1, +1, Q_0)$

$(Q_0, *, 1, +1, Q_0)$

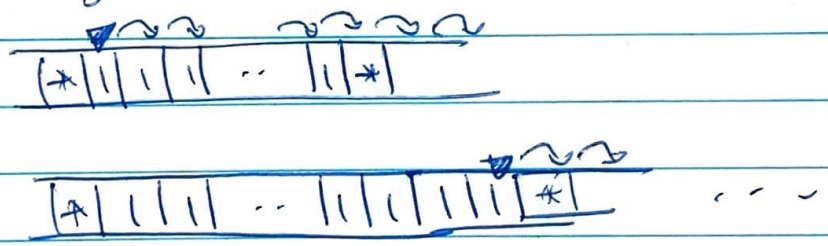
$(Q_1, a, a, +1, Q_1)$ for $a \in \{1, *\}$

always write 1 and go right

irrelevant

never reached

- Instructions say: on input $w = 11\cdots1$ keep printing 1's and proceed to right indefinitely (never go to halt state)
- So program never halts:



- we can use TMs to formalize notion of a decision problem being "decidable"

Def'n A decision problem (A, P) on an alphabet A is decidable if there is a TM M on a finite alphabet $B \supseteq A \cup \{Y, N\}$ s.t. for every word $w \in A^*$ we have:

$w \in P \Rightarrow$ on input w, M halts w/ output Y
$w \notin P \Rightarrow$ on input w, M halts w/ output N

In other words: M is an algorithm that determines unequivocally whether a given word w belongs to P or not.

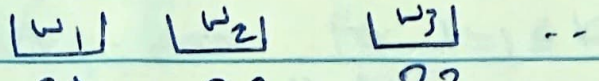
↳ if no such M, P is <u>undecidable</u>

Register machines:

- there are other formal models of a "computer" besides TMs
  ↳ we'll consider one called a "register machine"

- turns out: all of these models of computation are equivalent – this is the famous Church–Turing thesis (more in a moment)

- intuitively, a register machine (RM) on an alphabet A consists of:
  ↳ an infinite collection of registers $R1, R2, R3, \ldots$ each of which can store a word (of any length) from A^*.



↳ an ordered list of instructions (i.e. a program) that execute operations like

- delete word in nth register
- overwrite word in R_m w/ word in R_n
- go to a specific instruction in the list
- halt

etc...

Precisely:

Defn a register machine N on an alphabet $A = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ is a program consisting of an ordered list of instructions

$1: I_1$
$2: I_2$
$3: I_3$
$\vdots$
$n: I_n$

each of which is of one of the following types

ADD$_j$ R_m for $1 \le j \le k$, $m = 1, 2, 3, \ldots$
"add σ_j to end of word in R_m"

DEL R_m for $m = 1, 2, \ldots$
"delete leftmost symbol of word in R_m"

CLR R_m "delete word in R_m"

$R_p \leftarrow R_m$ "replace word in R_p w/ word in R_m"

GOTO$_\ell$ for $\ell = 1, 2, \ldots, n$
"go to the ℓth instruction $\ell : I_\ell$"

IF FIRST$_j$ R_m GOTO$_\ell$ "if the first symbol in word in R_m is σ_j go to ℓth instruction; otherwise go to next instruction"

STOP

- further we insist I_i = STOP iff $i = n$ (i.e. each program has exactly one stop instruction @ the end)

instructions run in order unless GOTO sends us elsewhere in list

↪ a computation of the machine runs as follows:

① on input $w \in A^*$ put w in first register R_1 (other registers empty) and run the program until we reach STOP (or indefinitely if STOP never reached)