Models of Computation

History: beginning of 20th century: people asking for "mechanical procedures" for solving math problems – e.g., solving integer polynomial equations

193x: Gödel, Church, Kleene, Turing: formal definitions of computation (all of them equivalent, yet different)

What problems we want to solve?

↳ fix "language"

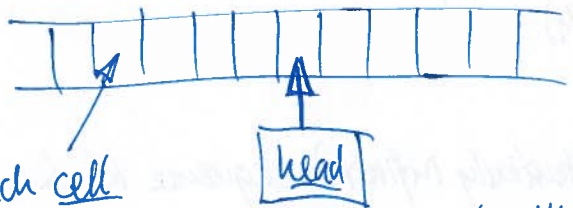
- Σ – finite alphabet of symbols (characters) – examples: $\{0,1\}$, $\{a \dots z\}$, math symbols
- Σ^* – set of all words (finite sequences) over Σ
  - ε ... empty word
  - $|\alpha|$ ... length
  - $\alpha\beta$ ... concatenation
  - symbol ≈ 1-symbol word
  - $\alpha[i]$ ... i-th symbol (starting with 0)
  - $\alpha[i:j] = \alpha[i] \dots \alpha[j-1]$ ... subword
  - $\alpha[:j] = \alpha[0:j)$ ... prefix
  - $\alpha[i:] = \alpha[i:|\alpha|]$ ... suffix

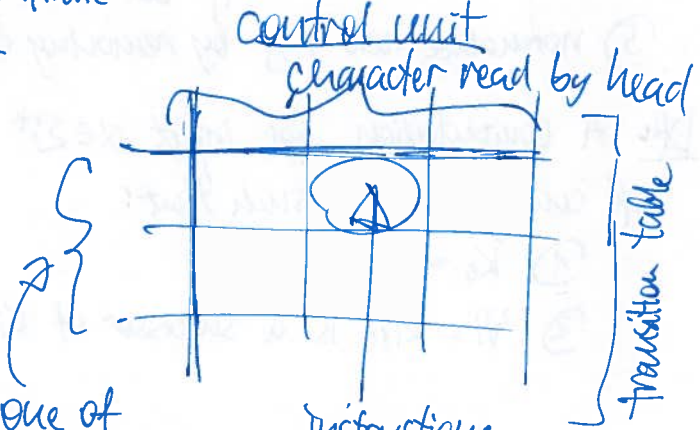
every alphabet can be encoded in every other (assuming $|\Sigma| \geq 2$)

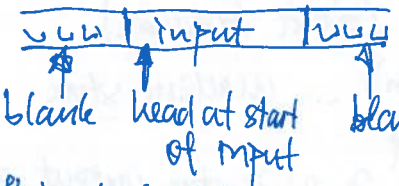
- problem: function from Σ^* to Σ^*
- decision problem: $f : \Sigma^* \to \{0,1\}$
  - ↳ also viewed as language $L \subseteq \Sigma^*$: $\alpha \in L \Leftrightarrow f(\alpha) = 1$ (characteristic function of a set)
- usually we find encoding of inputs (e.g. polynomials) to strings
  - concrete encoding doesn't matter (they can be converted algorithmically)
  - what happens if the input string is not a valid encoding?
    - ↳ suppose we always answer ε or 0 in such cases.

Turing Machines

motivation: a mathematician with finite mind working on an infinite blackboard





- start: 

& initial state

- finish: special state(s)

Now formally...

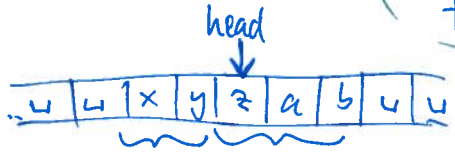
Df: A Turing machine consists of:

- Q ... a finite set of states
- $q_0 \in Q$ ... initial state
- $q_+, q_- \in Q$ ... final states (accepting & rejecting) } q_0, q_+, q_- all distinct
- Σ ... non-empty finite input alphabet
- $\Gamma \supseteq \Sigma$ ... finite work alphabet
- $\sqcup \in \Gamma \setminus \Sigma$ ... blank symbol
- $\delta : (Q \setminus \{q_+, q_-\}) \times \Gamma \to Q \times \Gamma \times \{\leftarrow, \bullet, \rightarrow\}$ ... transition function

Df: Configuration of the TM: $(q, \lambda, \varrho) \in Q \times \Gamma^* \times \Gamma^*$

- q: current state
- λ: left part of tape before the head with leading blanks trimmed
- ϱ: right part of tape from the head onwards with trailing blanks trimmed

↳ normalization



Example: ... ⊔ ⊔ 1 x y z a b ⊔ ⊔ ...

$\lambda\varrho$ = non-blank part of tape

on empty tape, all positions of the head are the same configuration

Df: A configuration (q, λ, ϱ), $q \neq q^+, q^-$ has a successor defined in this way:

1. extend λ by a leading ⊔, ϱ by a trailing ⊔
2. now, $\lambda = \lambda' y$, $\varrho = x\varrho'$ for some x, y, λ', ϱ'
3. evaluate $\delta(q, x)$ ... get (q', x', dir)
4. execute instruction: if $dir = \bullet$ ... $(q', \lambda, x'\varrho')$
   if $dir = \leftarrow$ ... $(q', \lambda', yx'\varrho)$
   if $dir = \rightarrow$ ... $(q', \lambda x', \varrho')$ } new config.
5. normalize new λ, ϱ by removing blanks

Df: A computation for input $\alpha \in \Sigma^*$ is a (potentially infinite) sequence $K_0, K_1, \ldots$ of configurations such that:

1. $K_0 = (q_0, \varepsilon, \alpha)$
2. $\forall i$: K_{i+1} is a successor of K_i (if K_{i+1} exists) ← therefore state q^+ or q^- never occurs except for the last config of a finite seq.
3. if seq. is infinite: the computation diverges (doesn't terminate)
   if K_n is last: ~~K_n contains~~ $K_n = (q_n, \lambda_n, \varrho_n)$ ... machine stops

q_n: comp. accepts or rejects the input ← q_+ or q_-

$\lambda_n\varrho_n$ is the output of computation

Df: Computability:

Function $f: \Sigma_1^* \to \Sigma_1^*$ is computable (also general recursive)
$\equiv \exists$ M Turing machine s.t.
$\forall \alpha \in \Sigma_1^*$ M(α) halts and outputs $f(\alpha)$
(M on input α & its computation)

Function $f: \Sigma_1^* \to \Sigma_1^* \cup \{\uparrow\}$ $(\uparrow \notin \Sigma_1)$ (divergence)
is partially computable (also partially recursive) $\equiv \exists$ M T.m.
s.t. $\forall \alpha \in \Sigma_1^*$: if $f(\alpha) = \uparrow$: M(α) diverges
else M(α) halts & outputs $f(\alpha)$

Language $L \subseteq \Sigma_1^*$ is computable (also recursive)
$\equiv \exists$ M T.m. s.t.
$\forall \alpha \in \Sigma_1^*$ M(α) always halts
& (accepts $\alpha \Leftrightarrow \alpha \in L$)
(accepts = ends in q^+)

↳ equivalent to char. fn of L computable

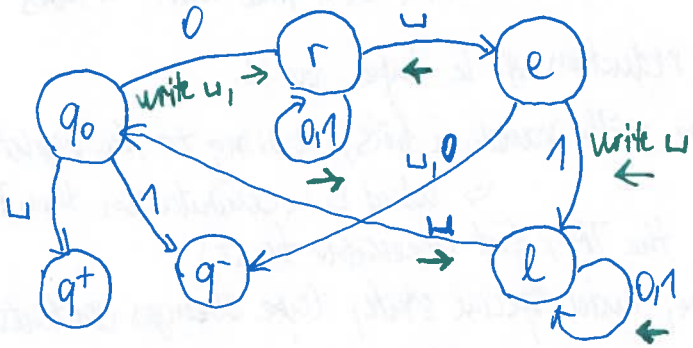
Language $L \subseteq \Sigma_1^*$ is partially computable (also recursively enumerable)
$\equiv \exists$ M T.m. s.t.
$\forall \alpha \in \Sigma_1^*$ M(α) halts $\Leftrightarrow \alpha \in L$.
(halts in state q^+)

↳ equivalent to $C'_L(\alpha) = \begin{cases} 1 & \text{if } \alpha \in L \\ \uparrow & \text{if } \alpha \notin L \end{cases}$ partially computable

Idea: Time & Space spent by computation
Time: # configurations visited ≅ # instructions executed
Space: # cells visited by the head

} will serve as basis for complexity theory (later)

Example: Recognizing $\{0^n 1^n\} \subseteq \{0,1\}^*$ by accepting / rejecting.
Idea: erase first 0 & final 1, repeat.

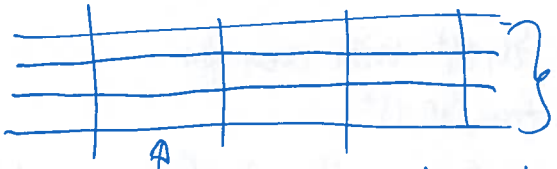


δ	0	1	␣
q_0	$(r, ␣, \rightarrow)$	$(q^-, ?, ?)$	$(q^+, ?, ?)$
r	$(r, 0, \rightarrow)$	$(r, 1, \rightarrow)$	$(e, ␣, \leftarrow)$
e	$(q^-, ?, ?)$	$(l, ␣, \leftarrow)$	$(q^-, ?, ?)$
l	$(l, 0, \leftarrow)$	$(l, 1, \leftarrow)$	$(q_0, ␣, \rightarrow)$

(doesn't matter)

$\Sigma_1 = \{0,1\}$
$\Gamma = \{0,1,␣\}$
$Q = \{q^0, q^+, q^-, r, e, l\}$

Ideas:
- Encode multiple variables with finite domains in the state – state is a tuple
- Multi-track tape

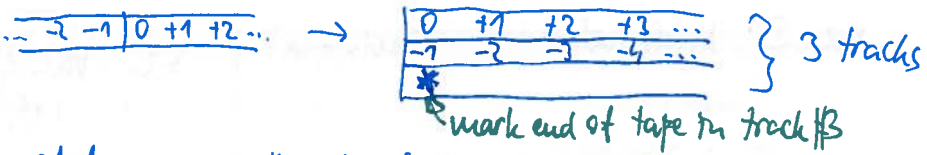


Head reads/writes k-tuples
But all tracks share head position

Remember to en/decode tape at start/end of computation.

Variants of the TM (robustness of definition)

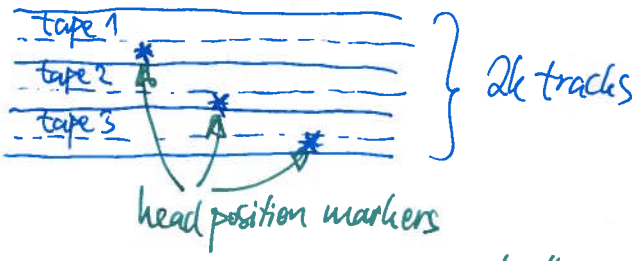
① One-way infinite tape ... 1-way → 2-way trivial
↳ equally powerful (set of computable functions remains unchanged...)
2-way → 1-way "fold tape in half" simulation



state contains "positive half-tape" switch.

② k tapes with independent heads (but sharing a common work alphabet (w.l.o.g.))

- transition function: $(Q \setminus \{q^+, q^-\}) \times \Gamma^k \to Q \times \Gamma^k \times \{\leftarrow, \rightarrow, \bullet\}^k$
- configuration, successor, computation easy to extend
  - start: tape #1 contains input, other tapes empty
  - end: tape #1 contains output
- Equally powerful ... k-tape → 1-tape:



1 step of orig. machine can be simulated by scanning the whole tape 2 times:
- find all heads, record symbols they see in state
- write symbols back & move heads

actually, you can do it in $\mathcal{O}(n \log n)$ time (exercise)

- But complexity changes: $\{0^n 1^n\}$ requires super-linear time with 1 tape but can be solved in $\mathcal{O}(n)$ time with 2 tapes
- Later: there is a more efficient reduction of k tapes to 2.

③ Randomized TM ... read-only tape with random bits, moving to the right only
↳ what is a computation then? ...

④ Oracles (functions defined outside the TM, but accessible to it)
- oracle tape: write query there, enter special state, tape changes contents to the answer

⑤ Interactive TM ... outside world can be modelled as an oracle 😉

⑥ Exercise: 2-Dimensional tape ... head moves $\leftarrow, \rightarrow, \uparrow, \downarrow, \bullet$

... in some sense, the most powerful physically feasible computer is a TM with 3-D tape (or maybe 2-D only to allow heat spreading...)

Exercises:
- accept strings in $\{0,1\}^*$ with even #1
- reverse a string from $\{0,1\}^*$
- add / multiply numbers written in binary ... what's the complexity?
  ↳ non-negative integers