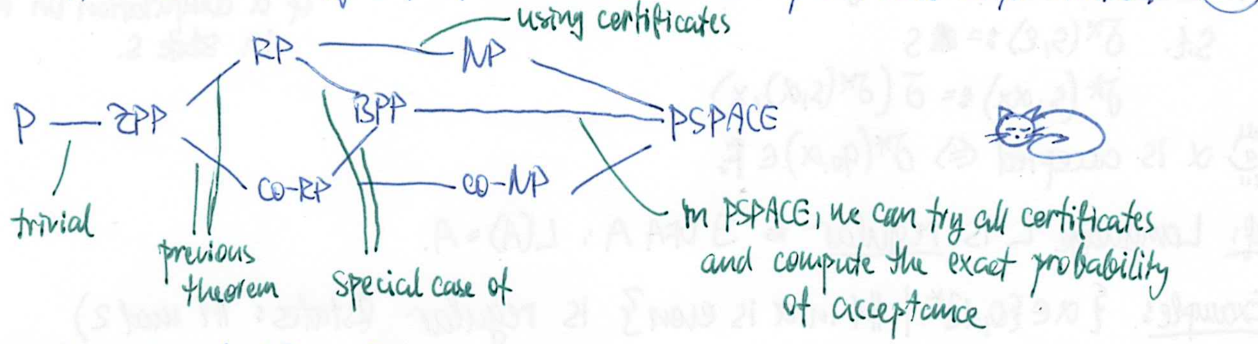
Exercise: What happens if we modify def. of BPP or RP to use expected time and/or MAYBE?

Inclusions:



Excercise: Define class PP: $L \in PP \equiv \exists M$ PTM running in w.c. time $O(poly(n))$ s.t. $P(L(\alpha) = M(\alpha)) > 1/2$ for all α.

beware: amplification doesn't work for PP (not exponentially!)

Show that: $NP \subseteq PP$, $co\text{-}NP \subseteq PP$, $BPP \subseteq PP$, $PP \subseteq PSPACE$.

Theorem: $BPP \subseteq P/poly$.

Proof: (Let $L \in BPP$) For inputs of size n: iterate $O(n)$ times to get $P(\text{error}) \leq \frac{1}{2} \cdot 2^{-n}$ (amplify)

Let $r :=$ # random bits used by the machine (certificate size)

For a fixed input α: $P_{\beta \in \{0,1\}^*}(V(\langle \alpha, \beta \rangle) \neq L(\alpha)) \leq \frac{1}{2} \cdot 2^{-n} \Rightarrow$ # "bad" certs for which this happens $\leq 2^r \cdot \frac{1}{2} \cdot 2^{-n}$

↳ taking union over all α: # bad certs $\leq 2^r \cdot \frac{1}{2} \cdot 2^{-n} \cdot 2^n = \frac{1}{2} \cdot 2^r < 2^r$

$\Rightarrow$ there exists a certificate which is good for all inputs: this will be the advice.

So our algorithm just calls V on $\langle$input, advice$\rangle$. This implies $L \in P/poly$.

Notes: It is known that $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$ (Sipser-Gács theorem) ← this is stronger than $BPP \subseteq PSPACE$.

There are no known BPP-complete problems nor hierarchy theorems. ← BPP is a "semantic" class, so diagonalization doesn't work

It's believed that BPP = P (otherwise hard-to-believe things happen)

REGULAR LANGUAGES

Df: Deterministic Finite-state Automaton (DFA) consists of:

- Q – a finite non-empty set of states
- Σ – a finite non-empty alphabet
- $\delta : Q \times \Sigma \to Q$ – transition function
- $q_0 \in Q$ – initial state
- $F \subseteq Q$ – a set of accepting states

Df: • Computation of a DFA over an input string $\alpha \in \Sigma^*$ is a sequence of states $s_0, s_1, \dots, s_{|\alpha|}$ ← uniquely determined such that $s_0 = q_0$ and $\forall i \; s_{i+1} = \delta(s_i, \alpha[i])$.

alternatively:
- DFA is a multi-graph with labelled edges (by Σ)
- computation is a walk in the graph starting in q_0 and labelled by the input α.

- The input is accepted $\equiv s_{|\alpha|} \in F$.
- $L(A)$ :: the language of all words accepted by the automaton A.

Df: Extended transition function $\delta^*: Q \times \Sigma^* \to Q$ ← $\delta^*(s,\alpha)$ is the final state of a computation on α starting in state s.

s.t. $\delta^*(s,\varepsilon) := s$

$\delta^*(s,\alpha x) := \delta(\delta^*(s,\alpha), x)$

α is accepted $\iff \delta^*(q_0,\alpha) \in F$.

Df: Language L is regular $\equiv \exists$ DFA A: $L(A) = A$.

Example: $\{\alpha \in \{0,1\}^* \mid \#1 \text{ in } \alpha \text{ is even}\}$ is regular (2 states: #1 mod 2)

Example: Every finite language is regular (states: prefixes of words in the language)

Example: $\{0^n 1^n \mid n \geq 0\}$ is not regular. If there existed a DFA accepting it: set $t := |Q|$, consider s_0 — s_t, where $s_i := \delta^*(q_0, 0^i)$. By Pigeon-hole principle, there is $i<j$ s.t. $s_i = s_j$. Now $\delta^*(q_0, 0^i 1^i) = \delta^*(q_0, 0^j 1^i)$, so $0^i 1^i$ is accepted $\iff 0^j 1^i$ is. ↯

Lemma (Pumping lemma for regular languages):

For every regular language L, there exists $n \geq 0$ such that:

Every $w \in L$, $|w| \geq n$ can be decomposed as $w = \alpha\beta\gamma$, where:

1. $\forall t \geq 0$ $\alpha\beta^t\gamma \in L$ (including $i=0$)
2. $\beta \neq \varepsilon$
3. $|\alpha\beta| \leq n$.

Proof: Consider an automaton accepting L. Set $n := |Q|$.

Given $w \in L$, $|w| \geq n$, define s_0 — s_m: $s_i := \delta^*(q_0, w[:i])$

let $m := |w|$

Since $m \geq n$, there is $i < j \leq n$ s.t. $s_i = s_j$.

Now set $\alpha := w[:i]$, $\beta := w[i:j]$, $\gamma := w[j:]$. ... this implies ② and ③

① $\delta^*(q_0,\alpha) = s_i = s_j = \delta^*(q_0,\alpha\beta)$ ... so $\delta^*(s_i,\beta) = s_i$, hence $\forall t \geq 0$ $\delta^*(q_0,\alpha\beta^t) = s_i$, so $\forall t$ $\delta^*(q_0,\alpha\beta^t\gamma)$ is always the same. For $t=1$, $\alpha\beta^t\gamma \in L$, so all $\alpha\beta^t\gamma \in L$.

Example: $0^n 1^n$ again ... If it were regular, use $0^n 1^n$ with n from the lemma. Both α,β must consist purely from 0s, so $\alpha\beta^t\gamma$ is $0^i 1^j$ and we can increase i, while staying inside the language. ↯

Lemma: Intersection of two regular languages (over a common alphabet) is regular.

Proof: Let L_1, L_2 be regular, DFA $A_1 = (Q_1, \Sigma, q_{01}, F_1)$ accepting L_1 and DFA $A_2 = (Q_2, \Sigma, q_{02}, F_2)$ accepting L_2.

Construct a product of A_1 and A_2:

$Q := Q_1 \times Q_2$

$\delta((s_1,s_2),x) := (\delta_1(s_1,x), \delta_2(s_2,x))$

$q_0 := (q_{01}, q_{02})$

$F := F_1 \times F_2$

we have $\delta^*((s_1,s_2),\alpha) = (\delta_1^*(s_1,\alpha), \delta_2^*(s_2,\alpha))$

so $\alpha \in L(A) \iff \alpha \in L_1 \cap L_2$.

Intuition: Run A_1, A_2 in parallel, accept iff both accepted.

Exercise: Regular languages are also closed under complement and ~~int~~ union.

Df: Non-deterministic Finite-state Automaton (NFA)

Like DFA, but $\delta: Q \times \Sigma_1 \to \mathcal{P}(Q)$ - we have multiple possible instructions to execute
and $Q_0 \subseteq Q$ replaces q_0 - multiple initial states

What changes: Computation requires $s_{i+1} \in \delta(s_i, \alpha[i])$, $s_0 \in Q_0$
There can be multiple computations for a given input, or perhaps none.
α is accepted $\equiv$ there exists a computation ending in an accepting state.

Df: $\delta^*: \mathcal{P}(Q) \times \Sigma_1^* \to \mathcal{P}(Q)$ defined as:
$\delta^*(S, \varepsilon) := S$, $\delta^*(S, \alpha x) := \bigcup_{t \in \delta^*(S,\alpha)} \delta(t, x)$.
} again: α is accepted $\Leftrightarrow \delta^*(Q_0, \alpha) \cap F \neq \emptyset$.

Thm: If L is accepted by an NFA, then it is regular. ← so non-determinism doesn't increase computing power of FAs

Proof: Construct a DFA $A'(Q', \Sigma_1, \delta', q_0', F')$ which simulates δ^* of the original NFA $A(Q, \Sigma_1, \delta, Q_0, F)$.
Let $Q' := \mathcal{P}(Q)$
$\delta'(s, x) := \delta^*(s, x)$ → Then $\delta'^*(q_0', \alpha) = \delta^*(Q_0, \alpha)$,
$q_0' := Q_0$ so $\alpha \in L(A') \Leftrightarrow \alpha \in L(A)$.
$F' := \{s \in Q \mid s \cap F \neq \emptyset\}$

Nicer generalization: ε-NFA, which adds ε-edges: these can be traversed without reading a symbol from the input

Df: Extend $\delta: Q \times (\Sigma_1 \cup \{\varepsilon\}) \to \mathcal{P}(Q)$.

Computation = walk from $q_0 \in Q_0$ s.t. concatenated edge labels yield input string.

Df: ε-closure $U_\varepsilon(s)$ of a state s := set of all states reachable from s using only ε-edges.
$U_\varepsilon(S)$ of $S \subseteq Q$:= $\bigcup_{s \in S} U_\varepsilon(s)$.

↳ extendending δ^* to ε-NFAs: $\delta^*(S, \varepsilon) := U_\varepsilon(S)$
$\delta^*(S, \alpha x) := U_\varepsilon\left(\bigcup_{t \in \delta^*(S,\alpha)} \delta(t, x)\right)$

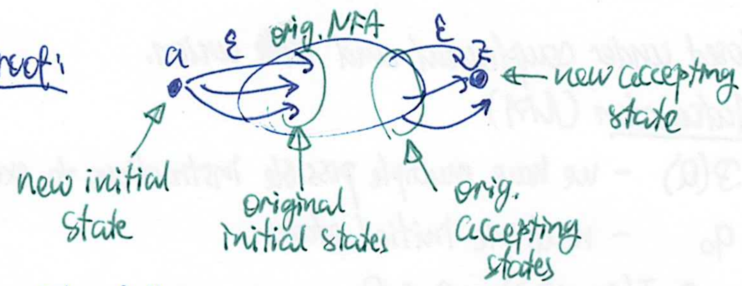
Thm: For every ε-NFA $A(Q, \Sigma_1, \delta, Q_0, F)$, there is a NFA $A' = (Q', \Sigma_1, \delta', Q_0', F')$ accepting the same language.

Proof: Just add ε-closure: $Q' := Q$
$Q_0' := U_\varepsilon(Q_0)$
$\delta'(S, x) := U_\varepsilon(\delta(S, x))$
$F' := F$
] so $\delta'^*(S, x) = \delta^*(S, x)$, hence $L(A) = L(A')$.

ε-NFAs accept still the same regular languages, but they are easier to construct.

Lemma: For every ε-NFA, there is an equivalent ε-NFA (accepting the same language) which has a unique initial state (with no incoming edges) and unique accepting state (with no outgoing edges)

Proof:

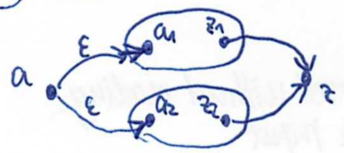


Theorem: The following operations with languages preserve regularity:

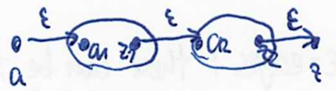
- $\bar{L}$ complement
- $L_1 \cap L_2$ intersection
- $L_1 \cup L_2$ union
- $L_1 \cdot L_2 := \{\alpha \cdot \beta \mid \alpha \in L_1, \beta \in L_2\}$ concatenation (associative)
- L^k: $L^0 := \{\varepsilon\}$, $L^{t+1} := L^t \cdot L$ power
- $L^* := \bigcup_{t \geq 0} L^t$ iteration
- $L^+ := \bigcup_{t > 0} L^t$ positive iteration
- $L^R := \{\alpha^R \mid \alpha \in L\}$ reversal — word written backwards

Proof: For $\bar{L}$ and $L_1 \cap L_2$, we already have the proof. Otherwise use ε-NFAs with unique init/acc. state.

① Union



② Concatenation



& hence also powers

③ Positive iteration

④ Iteration: add union with $\{\varepsilon\}$

⑤ reversal: swap role of a, z, switch orientation of all edges.

this is an equivalent definition of regularity which does not use automata

Theorem (Kleene): L is regular $\Leftrightarrow$ L can be constructed from $\emptyset$, $\{\varepsilon\}$, $\{x\}$ for $x \in \Sigma$, using finitely many unions, concatenations and iterations.

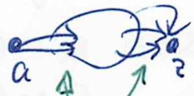
Proof: $\Leftarrow$ follows from the previous thm.

Prove $\Rightarrow$ using even more generalized NFAs, where each edge is labelled by a language and we can traverse the edge if we read a word in that language from the input.

Consider a DFA accepting L. We will transform it gradually to $a \xrightarrow{L} z$, always preserving the accepted language & making sure that languages on the edges can be constructed in the required way (using $\cup, \cdot, *$).

Steps:

⓪ initialization: add unique init. & acc. states:



ε-edges (labelled by language $\{\varepsilon\}$)

repeat while there are parallel edges

① elimination of parallel edges: replace $x \overset{L_1}{\underset{L_2}{\rightrightarrows}} y$ by $x \xrightarrow{L_1 \cup L_2} y$ (we can have $x = y$ here)

② elimination of states: remove a state $s \neq a, z$, routing around it:

a) if s has no loops: replace all $x \xrightarrow{L_1} s \xrightarrow{L_2} y$ by $x \xrightarrow{L_1 \cdot L_2} y$

b) if s has a loop: replace all $x \xrightarrow{L_1} s \xrightarrow{L_3} y$ by $x \xrightarrow{L_1 \cdot L_2^* \cdot L_3} y$

repeat until only a, z remain

Theorem: DSPACE(1) = NSPACE(1) = class of all regular languages.

Building the proof: Deterministic machines first.

1. TMs with just the input tape, which is read only & head doesn't move left ... this is equivalent to a DFA (technical detail: how do we accept/reject?)
2. Allow moving left. Tech. detail: delimit the input as $\langle\alpha\rangle$. On $\langle$, the TM must move right, on $\rangle$, it must move left.

[This is called the bi-directional DFA. We will prove that these accept just regular languages.
(Infinite loop / divergence is interpreted as rejecting the input.)

3. Allow work tapes of constant size: their contents & head positions can be moved inside machine state → this is equivalent to 2.
4. Non-deterministic TMs: 1 becomes NFA, so also regular
   2 will need a generalized proof
   3 still reduces to 2.

Need to prove: If L is accepted by a bi-dir. DFA, then L is regular.

Consider computation of the bi-dir. DFA on suffixes of a given input α:

- we start on $\alpha[i]$ in some state s
- we let the computation run until
  - it stops in q^+ or q^-
  - it diverges (equivalent to q^-)
  - it leaves the suffix $\alpha[i:]$ by moving left from position 1.
- we can describe this behavior by a function $f_i : Q \setminus \{q^0, q^-\} \to Q$

The f_i's can be constructed backwards ...

- $f_{|\alpha|}$ is trivial (the TM must ~~never~~ not move right, so iterate δ until it moves left/stops)
- $f_{i+1} \to f_i$: for $f_i(s)$, construct a sequence of states:

$s_0 = s$

$s_j \to s_{j+1}$: if $s_j = q^+/q^-$, stop & define $f_i(s) := s_j$

otherwise evaluate $\delta(s_j, \alpha[i]) \to (s_j', \text{movement})$

- if movement $= \leftarrow$: stop & define $f_i(s) := s_j'$
- if movement $= \bullet$: $s_{j+1} := s_j'$ & continue
- if movement $= \rightarrow$: $s_{j+1} := f_{i+1}(s_j')$ & continue

if $s_{j+1} = s_i$ for $i \le j$, the machine diverged, so $f_i(s) := q^-$ & stop.

$\Rightarrow f_i$ is a function of f_{i+1} and $\alpha[i]$.

So there is a DFA processing α^R, whose states are the f_i's.

α^R is accepted $\Leftrightarrow f_1(q_0) = q^+$

so L^R is regular, therefore L is also regular.

minor technicality: We let the TM start on $\langle$ instead of the first char. of α