

Languages and decision problems

An alphabet is a finite set.

Eg. $\Sigma = \{0,1\}$

A string over an alphabet Σ is a finite sequence from Σ .

Σ^* = the set of all strings (including the null string) over Σ .

A language over Σ is a set of strings over Σ .

Eg. *Prime* = $\{\alpha \in \Sigma^* \mid \alpha \text{ is the binary encoding of a prime number}\}$

A decision algorithm for a language L over Σ is an algorithm which on input a string $\alpha \in \Sigma^*$,
outputs "yes" if $\alpha \in L$;
outputs "no" if $\alpha \notin L$.

Assume standard binary encoding for natural numbers, rational numbers, graphs, etc.

$\langle \rangle$: "standard binary encoding"

Prime = $\{ \langle n \rangle \mid n \text{ is a prime number} \}$ captures the question of distinguishing primes from composites.

Turing machine -- formal model

Deterministic Turing machine (DTM)

$$M = (Q, \Sigma, \delta, q_0, q_f)$$

Q : a finite set of states

$$q_0, q_f \in Q$$

δ : transition function

$$\delta : Q - \{q_f\} \times \Sigma \rightarrow Q \times \Sigma \times \{L, R, S\}$$

Configuration : $\alpha q \beta$, $\alpha, \beta \in \Sigma^*$, $q \in Q$.

Transition

$$\delta(q, a) = (p, b, S) \Rightarrow \alpha qa\beta \xrightarrow{M} \alpha pb\beta$$

$$\delta(q, a) = (p, b, L) \Rightarrow \alpha cqa\beta \xrightarrow{M} \alpha pcb\beta$$

$$\delta(q, a) = (p, b, R) \Rightarrow \alpha qa\beta \xrightarrow{M} \alpha b p\beta$$

The language accepted by M

$$L(M) = \{ \alpha \in \Sigma^* \mid q_0 \alpha \xrightarrow{*} \beta q_f \gamma, \beta, \gamma \in \Sigma^* \}$$

steps := # transitions

$$C_0 \xrightarrow{M} C_1 \xrightarrow{M} \dots \xrightarrow{M} C_n$$

The function computed by a Turing machine M

$$f_M(\alpha) = \beta \text{ if } q_0 \alpha \xrightarrow{*} \beta_1 q_f \beta_2 \text{ on M where } \beta = \beta_1 \beta_2.$$

M decides a language L iff

$$f_M(\alpha) = 1 \text{ for all } \alpha \in L, \text{ and}$$

$$f_M(\alpha) = 0 \text{ for all } \alpha \notin L.$$

M accepts L in polynomial time if

$\exists c$, # transition steps for M to accept all

$\alpha \in L$ is $O(|\alpha|^c)$.

Turing machines for computing functions and deciding languages

The function computed by a Turing machine M

$f_M(\alpha) = \beta$ if $q_0\alpha \xrightarrow{*} \beta_1q_f\beta_2$ on M where $\beta = \beta_1\beta_2$.

f_M is *recursive* if $f_M(\alpha)$ is defined for all inputs α .

M decides a language L iff

$f_M(\alpha) = 1$ for all $\alpha \in L$, and

$f_M(\alpha) = 0$ for all $\alpha \notin L$.

Nondeterministic Turing machine (NDTM)

$$M = (Q, \Sigma, \delta, q_0, q_f)$$

Q : a finite set of states

$$q_0, q_f \in Q$$

δ : transition function

$$\delta : Q \times \Sigma \rightarrow 2^{Q \times \Sigma \times \{L, R, S\}}$$

$$\delta(q, a) = \{(p_i, b_i, D_i), i = 1, \dots, l\} \quad D_i = L, R, \text{ or } S$$

or the empty set.

$$\text{Eg. } \delta(p, 1) = \{(q, 0, L), (r, 1, S)\}$$

$$001p1 \xrightarrow{M} 00q10$$

$$\text{or, } 001p1 \xrightarrow{M} 001r1$$

$$(p, b, S) \in \delta(q, a) \Rightarrow \alpha q a \beta \xrightarrow{M} \alpha p b \beta$$

$$(p, b, L) \in \delta(q, a) \Rightarrow \alpha c q a \beta \xrightarrow{M} \alpha p c b \beta$$

$$(p, b, R) \in \delta(q, a) \Rightarrow \alpha q a \beta \xrightarrow{M} \alpha b p \beta$$

The language accepted by M

$$L(M) = \{ \alpha \in \Sigma^* \mid q_0 \alpha \xrightarrow{*} \beta q_f \gamma, \beta, \gamma \in \Sigma^* \}$$

#steps := # transitions

$$C_0 \xrightarrow{M} C_1 \xrightarrow{M} \dots \xrightarrow{M} C_n$$

M accepts L in non-deterministic polynomial time if

$\exists c$, for all $\alpha \in L$, there is a transition sequence of

$O(|\alpha|^c)$ steps for M to accept α .

P vs NP

A language L is in P iff there is a DTM which accepts L in polynomial time.

A language L is in NP iff there is a NDTM which accepts L in polynomial time. Equivalently, L is in NP iff L has a deterministic polynomial time verifier; that is a polynomial time algorithm $V(x,y)$ such that

$\exists c$, for all x , $x \in L$ iff $\exists y, |y| \leq |x|^c, V(x, y) = 1$.

NP models poly-time verification

Satisfiability : Given a Boolean formula $\varphi(x_1, \dots, x_n)$,
to decide if φ is satisfiable.

NP Algorithm :

Guess a truth assignment $t : \{x_1, \dots, x_n\} \rightarrow \{0,1\}$

If $\varphi(t(x_1), \dots, t(x_n)) = 1$, output "yes".

Corresponding poly - time verification algorithm

Input : $\varphi(x_1, \dots, x_n)$, $\tau \in \{0,1\}^n$

If $\varphi(\tau(1), \dots, \tau(n)) = 1$ output "yes"

Remark : $\text{yes} \Rightarrow \tau$ is a certificate for the satisfiability of φ

Example

Clique : Given a graph $G = (V, E)$ and an integer k ,
to decide if G has a k - clique. Assume $V = \{1, \dots, n\}$

NP Algorithm :

Guess k vertices ($t : \{1, \dots, n\} \rightarrow \{0, 1\}$)

If these k vertices form a clique, output "yes"

Corresponding poly - time verifier

Input : $G = (V, E), k, \tau \in \{0, 1\}^k$, where $V = \{1, \dots, n\}$

If $\{i \mid \tau(i) = 1\}$ forms a clique, output "yes".

Remark : $\text{yes} \Rightarrow \tau$ is a certificate for G .

A language L is in NP iff L has a polynomial time verifier.

Poly - time verifier \Rightarrow in NP.

Suppose L has a poly - time verifier :

$V(x, y)$ such that $x \in L$ iff $\exists y, |y| \leq |x|^c, V(x, y) = 1$.

Corresponding NP - algorithm for L :

On input x

Guess a string y of length $\leq |x|^c$.

Call $V(x, y)$.

Polynomial time transformation (reduction)

Let L_1 and L_2 be two languages over Σ_i , $i = 1, 2$, resp.

Suppose there is a poly - time algorithm T such that

on input $x \in \Sigma_1^*$, Algorithm T outputs a $y \in \Sigma_2^*$ such that

$$x \in L_1 \Leftrightarrow y \in L_2 .$$

(T is therefore a polynomial time computable function from Σ_1^* to Σ_2^* .)

Then we say that L_1 is polynomial time reducible to L_2 .

Write $L_1 \leq_P L_2$.

Satisfiability problem

A literal is either a Boolean variable or the negation of a variable.

A clause is the disjunction (\vee) of literals.

A Boolean formula in conjunctive normal form (CNF) is the conjunction (\wedge) of clauses. It is a k -CNF if each clause has k literals.

Eg. $(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$

3-CNF : Given a Boolean formula in 3-CNF, to decide if it is satisfiable.

Reduction from search to decision

Suppose \exists poly - time algorithm D for CNF :

$$D(\varphi(x_1, \dots, x_n)) = 1 \Leftrightarrow \varphi \in \text{CNF}.$$

Algorithm S finds a satisfying assignment for $\varphi \in \text{CNF}$

if $D(\varphi(1, x_2, \dots, x_n)) = 1$, set $x_1 = 1$ and call

$S(\varphi(1, x_2, \dots, x_n))$ recursively; otherwise

set $x_1 = 0$ and call $S(\varphi(0, x_2, \dots, x_n))$ recursively.

Time : $O(nT_D(\|\varphi\|))$

NP-completeness

A language L is NP-complete if

- L is in NP.
- For all $L' \in \text{NP}$, $L' \leq_p L$.

Proof that 3-CNF is NP-complete.

- A generic reduction from any NP problem to 3-CNF: Given any $L \in \text{NP}$, $L \leq_p \text{CNF}$.
- CNF is capable of expressing any NP computation.

Suppose $L \in NP$, accepted by some NDTM M in $p(n)$ time.

Demonstrate a poly - time algorithm T which

on input a string w , constructs a CNF ϕ_w so that

$w \in L \Leftrightarrow \phi_w$ is satisfiable (hence \in CNF).

$|\phi_w|$ is polynomial in $|w|$.

Consequently, if $CNF \in P$, then $L \in P$:

Suppose a p - time algorithm A solves the CNF problem

in $O(|\phi|^c)$ time.

$|\phi_w| = O(|w|^d)$.

To decide if $w \in L$,

$T(w)$ outputs ϕ_w .

$A(\phi_w)$. --- time $O(|\phi_w|^c) = O(|w|^{cd})$.

Polynomial time transformation

L accepted by a NDTM M in $p(n)$ time.

$$M = (Q, \Sigma, q_0, q_f, \delta)$$

$$Q : q_0, \dots, q_f$$

$$\Sigma = X_1, \dots, X_m$$

Variables of ϕ_w and intended meaning :

[cells] $C(i, j, t)$ $1 \Leftrightarrow$ at time t , cell i contains X_j

[state] $S(i, t)$ $1 \Leftrightarrow$ at time t , M is in state q_i

[head] $H(i, t)$ $1 \Leftrightarrow$ at time t , head of M reading cell i

$$n = |w|, 1 \leq i \leq p(n), 0 \leq t \leq p(n), 1 \leq j \leq m.$$

A useful Boolean expression

$$U(x_1, \dots, x_r) := (x_1 \vee \dots \vee x_r) \wedge_{i \neq j} (\neg x_i \vee \neg x_j) \text{ -- 1 iff exactly one of the var is 1.}$$

Expressing Head position :

$$A_t = U(H(1, t), \dots, H(p(n), t))$$

[At time t, head is reading exactly one cell]

$$A = \wedge A_t, 0 \leq t \leq p(n)$$

Cell contents

$$B_{i,t} = U(C(i, 1, t), \dots, C(i, m, t))$$

[At time t, cell i contains exactly one symbol]

$$B = \wedge B_{i,t}$$

States :

$$C = \wedge_t U(S(0, t), \dots, S(f, t))$$

[At time t, M is in exactly one state]

Expressing transitions

$E_{i,j,k,t}$: if at time t , cell i contains X_j , then one rule
in $\delta(q_k, X_j)$ is applied.

$$E_{i,j,k,t} = \neg C(i, j, t) \vee \neg H(i, t) \vee \neg S(k, t) \vee \Delta(i, j, k, t)$$

$$\Delta(i, j, k, t) = \bigvee_z \tau(i, j, k, t, z)$$

Say $\delta(q_k, X_j) = \{(q_1, X_1, R), \dots, (q_m, X_m, D_m)\}$

$$\tau(i, j, k, t, 1) = C(i, 1, t+1) S(1, t+1) H(i+1, t+1) :$$

at time $t+1$, cell i contains X_1 , M is in state q_1 , and head is reading cell $i+1$.

$F = S(f, p(n))$ [in accepting state by time $p(n)$]

$G = S(0,0)H(1,0) \bigwedge_{i=1}^{p(n)} C(i, w_i, 0),$

$w = w_1, \dots, w_n, \mathbf{b}, \dots, \mathbf{b}$ [Initial configuration]

$\phi_w = A \wedge B \wedge C \wedge E \wedge F \wedge G$

If $w \in L$, assign the variables according to an accepting transition sequence $\leq p(n)$ steps. $\Rightarrow \phi_w$ is satisfied.

If ϕ_w is satisfiable, a satisfying assignment on the var. translates directly into an accepting seq. of transition of length $\leq p(n)$.

More NP-complete problems

Thm Suppose $L_1 \leq_P L_2$. Then $L_2 \in P \Rightarrow L_1 \in P$.

Thm. Suppose $L \in \text{NP}$ and $L' \leq_P L$.

If L' is NP - complete, then L is also NP - complete.

Lemma 3 - $\text{CNF} \leq_P \text{Clique}$

Lemma $\text{Clique} \leq_P \text{Vertex Cover}$

Cor. Clique is NP - complete.

Cor. Vertex - Cover is NP - complete.

More NP-complete problems

Independent - set

$$G = (V, E)$$

$I \subseteq V$ is independent if for all $u, v \in I, (u, v) \notin E$.

IS: Given a graph $G = (V, E)$ and an integer k , to decide if G has an independent set of k vertices.

Hamiltonian cycle: Given a graph $G = (V, E)$ to decide if there is a simple cycle on G that contains every vertex.

Traveling salesman problem: Given a graph $G = (V, E)$ with weight $w: E \rightarrow \mathbf{Z}_{\{\geq 0\}}$ and an integer k , to decide if G has a Hamiltonian cycle of weight no greater than k .

Integer linear programming : Given an integer matrix

$$A = (a_{ij})_{m \times n} \text{ and vector } b,$$

to decide if there is an integer solution to

$$Ax = b, x \geq 0.$$

(0,1) – ILP : Given an integer matrix

$$A = (a_{ij})_{m \times n} \text{ and vector } b,$$

to decide if there is an integer solution to

$$Ax = b, x \in \{0,1\}.$$