Lesson 2: NP-complete languages

Theme: The notion of **NP**-completeness and **coNP**-completeness.

1 Alternative definitions of the class NP

Recall that for a string w, the length of w is denoted by $|w|$. In the previous lesson, we define the class **NP** as follows.

Definition 2.1 A language L is in **NP** if there is $f(n) = \mathsf{poly}(n)$ and an NTM $\mathcal{M}$ such that $L(\mathcal{M}) = L$ and $\mathcal{M}$ runs in time $O(f(n))$.

Definition 2.2 below is an alternative definition of **NP**.

Definition 2.2 A language $L \subseteq \Sigma^*$ is in **NP** if there is a language $K \subseteq \Sigma^* \cdot \{\#\} \cdot \Sigma^*$, where $\# \notin \Sigma$, such that the following holds.

- For every $w \in \Sigma^*$, $w \in L$ if and only if there is $v \in \Sigma^*$ such that $w\#v \in K$.
- There is $f(n) = \mathsf{poly}(n)$ such that for every $w\#v \in K$, $|v| \leqslant f(|w|)$.
- The language K is accepted by a polynomial time DTM.

For $w\#v \in K$, the string v is called the *certificate/witness* for w. We call the language K the *certificate/witness language* for L.

Indeed Def. 2.1 and 2.2 are equivalent. That is, for every language L, L is in **NP** in the sense of Def. 2.1 if and only if L is in **NP** in the sense of Def. 2.2.

2 NP-complete languages

Recall that a DTM $\mathcal{M}$ computes a function $F : \Sigma^* \to \Sigma^*$ in time $O(g(n))$, if there is a constant $c > 0$ such that on every word w, $\mathcal{M}$ computes $F(w)$ in time $\leqslant cg(|w|)$. If $g(n) = \mathsf{poly}(n)$, such function F is called *polynomial time computable* function. Moreover, if $\mathcal{M}$ uses only logarithmic space, it is called *logarithmic space computable* function. See Appendix A for more details.

The following definition is one of the most important definitions in computer science.

Definition 2.3 A language L_1 is *polynomial time reducible* to another language L_2, denoted by $L_1 \leqslant_p L_2$, if there is a polynomial time computable function F such that for every $w \in \Sigma^*$:

$$w \in L_1 \quad \text{if and only if} \quad F(w) \in L_2$$

Such function F is called polynomial time reduction, also known as *Karp reduction.*

If F is logarithmic space computable function, we say that L_1 is *log-space reducible* to L_2, denoted by $L_1 \leqslant_{\log} L_2$.

If L_1 and L_2 are in **NP** with certificate languages K_1 and K_2, respectively, we say that F is *parsimonious*, if for every $w \in \Sigma^*$, w has the same number of certificates in K_1 as $F(w)$ in K_2.

Definition 2.4 Let L be a language.

- L is **NP**-*hard*, if for every $L' \in \mathbf{NP}$, $L' \leqslant_p L$.
- L is **NP**-*complete*, if $L \in \mathbf{NP}$ and L is **NP**-hard.

Recall that a propositional formula (Boolean expression) with variables $x_1, \ldots, x_n$ is in Conjunctive Normal Form (CNF), if it is of the form: $\bigwedge_i \bigvee_j \ \ell_{i,j}$ where each $\ell_{i,j}$ is a literal, i.e., a variable x_k or its negation $\neg x_k$. It is in 3-CNF, if it is of the form $\bigwedge_i \ (\ell_{i,1} \ \vee \ \ell_{i,2} \ \vee \ \ell_{i,3})$. A formula φ is satisfiable, if there is an assignment of Boolean values true or false to each variable in φ that evaluates to true.

SAT	
Input:	A propositional formula φ in CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

Obviously, SAT can be viewed as a language, i.e., SAT $\stackrel{\text{def}}{=} \{\varphi : \varphi \text{ is satisfiable CNF formula}\}$.

Theorem 2.5 (Cook 1971, Levin 1973) SAT *is* **NP**-*complete.*

Proof. We have to show that SAT $\in$ **NP** and SAT is **NP**-hard. We first show that SAT $\in$ **NP**. Consider the following non-deterministic algorithm that decides SAT. On input formula φ, do the following.

- Let $x_1, \ldots, x_n$ be the variables in φ.
- For each $i = 1, \ldots, n$ do:
  - Non-deterministically assign the value of x_i to either true or false.
- Check if the formula φ evaluates to true under the assignment.
- If the formula evaluates to true, then ACCEPT.

  If the formula evaluates to false, then REJECT.

It is not difficult to show that the algorithm above accepts a formula φ if and only if it is satisfiable. This completes the proof that SAT $\in$ **NP**.

Now we show that SAT is **NP**-hard. That is, for every $L \in \mathbf{NP}$, $L \leqslant_p$ SAT.

Let $L \in \mathbf{NP}$. Let $\mathcal{M} = \langle \Sigma, \Gamma, Q, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}, \delta \rangle$ be the NTM that decides L in time $f(n) = \mathsf{poly}(n)$, where Σ is the input alphabet, Γ is the tape alphabet, Q is the set of states, q_0 is the initial state, q_{acc} is the accepting state, q_{rej} is the rejecting state and δ is the set of transitions. We denote by $\sqcup$ the blank symbol. We may assume that $\mathcal{M}$ has only 1 tape.

We will describe a deterministic algorithm $\mathcal{A}$ such that on every word w, it output a CNF formula φ such that the following holds.

$$w \in L \quad \text{if and only if} \quad \varphi \text{ is satisfiable.}$$

Intuitively, φ "describes" the accepting run of $\mathcal{M}$ on w such that it is satisfiable if and only if there is an accepting run of $\mathcal{M}$ on w. Let $n = |w|$. See Figure 1.

To describe the run, it uses the following boolean variables for every $q \in Q$, for every $\sigma \in \Gamma$, for every $1 \leqslant i, j \leqslant f(|w|)$:

$$X_{q,\sigma,i,j} \qquad \text{and} \qquad X_{\sigma,i,j}$$

Intuitively, $X_{q,\sigma,i,j}$ is true if and only if in step-j the head is in cell-i reading symbol σ and the TM is in state q; and $X_{\sigma,i,j}$ is true if and only if in step-j the content of cell-i is σ.

Essentially the formula φ states the following.

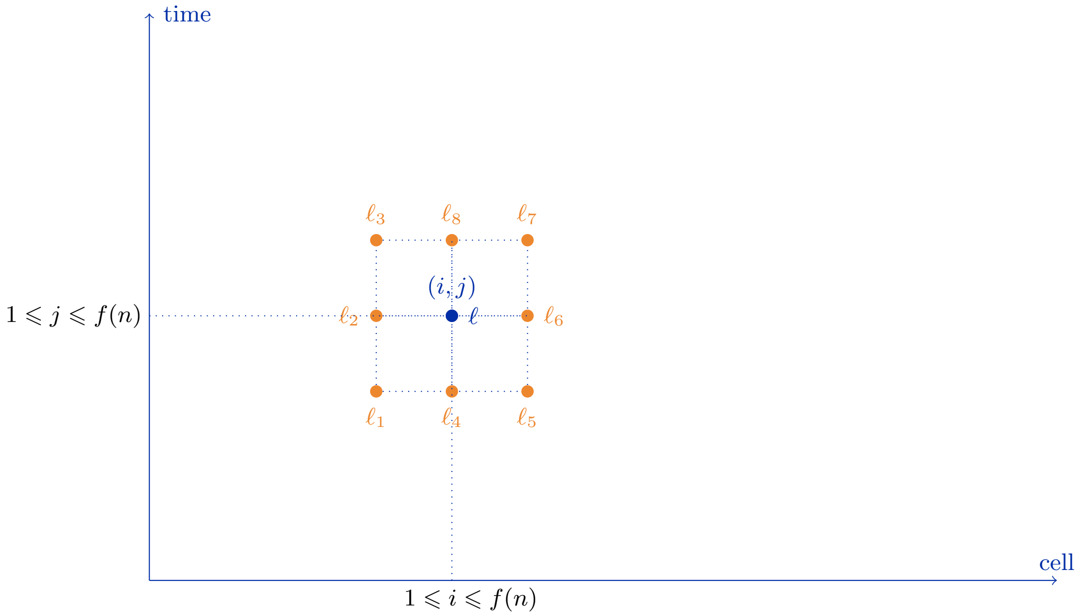

Figure 1: Each point (i, j) is labeled with a symbol $\ell \in (Q \times \Gamma) \cup \Gamma$. If $\ell = (q, \sigma) \in Q \times \Gamma$, it means in time-$j$ the NTM $\mathcal{M}$ is in state q and the head is in cell-i reading symbol σ. If $\ell = \sigma \in \Gamma$, it means in time-j the content of cell-i is σ. The labels ℓ and those in the neighboring points $\ell_1, \ldots, \ell_8$ must obey the transitions in of the NTM $\mathcal{M}$.

- In time-1 the labels of the points $(1, 1), \ldots, (1, f(n))$ is the initial configuration. It can be expressed as the following formula.

$$X_{q_0,a_1} \wedge X_{a_2} \wedge \cdots \wedge X_{a_n} \wedge \bigwedge_{i=n+1}^{f(n)} X_{\sqcup} \tag{1}$$

- The accepting state must appear somewhere. It can be expressed as the following formula.

$$\bigvee_{1 \leqslant i,j \leqslant f(n)} \bigvee_{\sigma \in \Gamma} X_{q_{\mathsf{acc}},\sigma,i,j} \tag{2}$$

- For every $1 \leqslant i, j \leqslant f(n)$, the labels in $(i-1, j), (i, j), (i+1, j+1)$ and the labels in $(i-1, j+1), (i, j+1), (i+1, j+1)$ must obey the transitions in $\mathcal{M}$.

  For example, if $(q, \sigma) \to (p, \alpha, \text{left})$ and $(q, \sigma) \to (r, \beta, \text{right})$ are transitions in $\mathcal{M}$, then the formula states the following.

$$\begin{aligned} &\bigwedge_{1 \leqslant i,j \leqslant f(n)} \ \bigwedge_{\sigma_1,\sigma_2,\sigma_3 \in \Gamma} X_{\sigma_1,i-1,j} \wedge X_{q,\sigma_2,i,j} \wedge X_{\sigma_3,i+1,j} \\ &\qquad\qquad \to \left(\begin{array}{c} (X_{p,\sigma_1,i-1,j+1} \wedge X_{\alpha,i,j+1} \wedge X_{\sigma_3,i+1,j+1}) \\ \vee \\ (X_{\sigma_1,i-1,j+1} \wedge X_{\beta,i,j+1} \wedge X_{r,\sigma_3,i+1,j+1}) \end{array} \right) \end{aligned} \tag{3}$$

- For every time j, there is exactly one i such that the label of (i, j) is of the form $(q, \sigma) \in$

$Q \times \Gamma$. It can be expressed as the following formula.

$$\bigwedge_{p,q\in Q \text{ and } \sigma,\sigma'\in\Gamma} \ \bigwedge_{1\leqslant j\leqslant f(n)} \ \bigwedge_{1\leqslant i<i'\leqslant f(n)} X_{q,\sigma,i,j} \to \neg X_{p,\sigma',i',j} \tag{4}$$

$$\wedge \qquad \bigwedge_{1\leqslant j\leqslant f(n)} \ \bigvee_{q\in\Sigma} \ \bigvee_{\sigma\in\Gamma} \ \bigvee_{1\leqslant i\leqslant f(n)} X_{q,\sigma,i,j} \tag{5}$$

The formula (4) states that there is at most one head and the formula (5) states that there is at least one head.

Formally, the algorithm $\mathcal{A}$ works as follows. On input w, it outputs the formula φ which is the conjunction of the formulas (1)– (5). It is not difficult to show that $w \in L$ if and only if φ is satisfiable. ∎

Remark 2.6 We note that in the proof of Theorem 2.5, the formula φ produced in the reduction from L to SAT satisfies the following.

$$\text{The number of accepting run of } \mathcal{M} \text{ on } w \ = \ \text{The number of satisfying assignment of } \varphi$$

Thus, the reduction in Theorem 2.5 is parsimonious.

Remark 2.7 There are two ways to show that a language L is **NP**-hard.

- The first is by definition, i.e., we show that for every language $K \in \mathbf{NP}$, there is a polynomial time reduction from K to L.
- The second is by choosing an appropriate **NP**-hard language, say SAT, and show that there is a polynomial time reduction from SAT to L.

3-SAT	
Input:	A propositional formula φ in 3-CNF.
Task:	Output true, if φ is satisfiable. Otherwise, output false.

Note that we can also view 3-SAT as the language 3-SAT $\stackrel{\text{def}}{=} \{\varphi : \varphi \text{ is satisfiable 3-CNF formula}\}$.

Theorem 2.8 3-SAT *is* **NP**-*complete.*

Proof. That is 3-SAT is in **NP** follows immediately from Theorem 2.5. To show that it is **NP**-hard, we reduce it from SAT. On input a CNF formula φ, if it has a clause of length greater than 3:

$$\ell_1 \vee \cdots \vee \ell_k \qquad\qquad \text{where } k \geqslant 4$$

split it into two clauses, where z is a new variable:

$$(\ell_1 \vee \cdots \vee \ell_{\lfloor k/2 \rfloor} \vee z) \ \wedge \ (\ell_{\lfloor k/2 \rfloor+1} \vee \cdots \vee \ell_k \vee \neg z)$$

Repeat it on each clause of length $\geqslant 4$ until we get 3-CNF. ∎

3 More NP-complete problems

We need a few terminologies. Let $G = (V, E)$ be a (undirected) graph.

- G is 3-colorable, if we can color the vertices in G with 3 colors (every vertex must be colored with one color) such that no two adjacent vertices have the same color.
- A set $C \subseteq V$ is a clique in G, if every pair of vertices in C are adjacent.
- A set $W \subseteq V$ is a vertex cover, if every edge in E is adjacent to at least one vertex in W.
- A set $I \subseteq V$ is independent, if every pair of vertices in I are non-adjacent.
- A set $D \subseteq V$ is dominating, if every vertex in V is adjacent to at least one vertex in D.

All the following problems are **NP**-complete.

3-COL

Input:	A (undirected) graph $G = (V, E)$.
Task:	Output true, if G is 3-colorable. Otherwise, output false.

CLIQUE

Input:	A (undirected) graph $G = (V, E)$ and an integer $k \geqslant 0$ in binary form.
Task:	Output true, if G has a clique of size $\geqslant k$. Otherwise, output false.

IND-SET

Input:	A (undirected) graph $G = (V, E)$ and an integer $k \geqslant 0$ in binary form.
Task:	Output true, if G has an independent set of size $\geqslant k$. Otherwise, output false.

VERT-COVER

Input:	A (undirected) graph $G = (V, E)$ and an integer $k \geqslant 0$ in binary form.
Task:	Output true, if G has a vertex cover of size $\leqslant k$. Otherwise, output false.

DOM-SET

Input:	A (undirected) graph $G = (V, E)$ and an integer $k \geqslant 0$ in binary form.
Task:	Output true, if G has an dominating set of size $\leqslant k$. Otherwise, output false.

4 coNP-complete problems

Analogous to **NP**-complete, we can also define **coNP**-complete problems.

Definition 2.9 Let K be a language.

- K is **coNP**-*hard*, if for every $L \in$ **coNP**, $L \leqslant_p K$.
- K is **coNP**-*complete*, if $K \in$ **coNP** and K is **coNP**-hard.

Theorem 2.10 *For every language K over the alphabet Σ, K is* **NP***-complete if and only if its complement $\overline{K}$ is* **coNP***-complete, where $\overline{K} \stackrel{def}{=} \Sigma^* - K$.*

Corollary 2.11 $\overline{\mathsf{SAT}} \stackrel{def}{=} \{\varphi : \varphi$ *is not satisfiable*$\}$ *is* **coNP***-complete.*

APPENDIX

A The notion of computable functions

Polynomial time computable functions. Let $F : \Sigma^* \to \Sigma^*$ be a function from Σ^* to Σ^*. Let $\mathcal{M}$ be a 2-tape DTM.

- $\mathcal{M}$ *computes the function* F, if $\mathcal{M}$ accepts every word $w \in \Sigma^*$ and when it halts, the content of its second tape is $F(w)$.
- $\mathcal{M}$ *computes* F *in time* $O(g(n))$, if there is a constant $c > 0$ such that on every word w, $\mathcal{M}$ decides w in time $c \cdot g(|w|)$.
- $\mathcal{M}$ *computes* F *in polynomial time*, if $\mathcal{M}$ computes F in time $O(g(n))$ for some $g(n) = \mathsf{poly}(n)$.
- F *is computable in polynomial time*, if there is a DTM $\mathcal{M}$ that computes F in polynomial time.

Logarithmic space computable function. A function $F : \Sigma^* \to \Sigma^*$ is computable in logarithmic space, if there is a 3-tape DTM $\mathcal{M}$ and a constant c such that on every $w \in \Sigma^*$ the following holds.

- $\mathcal{M}$ accepts w.
- $\mathcal{M}$ never change the content of tape-1, i.e., tape-1 always contains the input word w.

  In other words, tape-1 is "read-only" tape.
- $\mathcal{M}$ only uses at most $c \log |w|$ cells in tape-2.
- Tape-3 is "write-only" tape, i.e., the head in tape-3 can only write and move right.
- When $\mathcal{M}$ halts, the content of tape-3 is $F(w)$.