Formal Languages and Automata Theory IV

Rana Barua

Visiting Scientist
IAI, TCG CREST, Kolkata

1 Turing Machines

Turing machine (TM) was introduced by Alan Turing as a model of "any possible computation". It consists of an infinite tape with cells, a finite control and a tape-head scanning the content of a cell at any given instant. Depending on the state of the finite control and the letter being scanned by the tape-head, the the TM in one move can print a symbol in place of the scanned letter, move the tape-head one cell to the right or left and enters, perhaps, a new state. Initially, the input string w is placed in consecutive cells with all other cells occupied by the **blank** symbol B.The TM starts at an initial state scanning the first letter of the input string w. If after finitely many moves, the TM enters an accepting state then the TM accepts the input w. Formally,

Definition 1. *A Turing machine $\mathcal{M}$ is a 7-tuple $(\Sigma, \Gamma, Q, q_0, \delta, B, F)$, where*

1. $\Sigma \subseteq \Gamma$*: is the* ***input alphabet*** *not containing* B*,*
2. Γ*: is the* ***tape alphabet****,*
3. Q*: is the finite set of* ***states****,*
4. $q_0 \in Q$*: is the* ***initial state****,*
5. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$*, is the* ***transition function****,*
6. $B \in \Gamma$*: is the* ***blank symbol****,*
7. $F \subseteq Q$*: is the set of* ***accepting*** *or* ***final states****.*

Instantaneous Description(ID): Gives the configuration of a TM at any given instant. It should give information about

1. the state of the Turing machine
2. the contents of the tape and
3. the position of the tape-head.

Hence, it is encoded by

$$w_1 q w_2$$

with the TM in state q scanning the first letter of w_2. Here w_1 is the portion of the string to the left of the tape-head starting from the first non-blank symbol, while w_2 is the portion of the string staring from the symbol being scanned by the tape-head till the last non-blank symbol. If the first/last non-blank symbol is to the right/left of the tape-head, then w_1/w_2 is λ/B.
For instance, the configuration

$\leftarrow | B | a_1 | B | a_2 | a_3 | B \rightarrow$
$- - - - - - - - - - \uparrow - - - - - - - - - -$
$. \qquad\qquad\qquad\qquad |q|$

is encoded by $a_1 B a_2 q a_3$.
Initial configuration or ID: Clearly, the starting configuration/ID is $q_0 w$, where w is the input string.

Final or accepting ID: If a configuration contains an accepting state then the ID is an accepting or final ID. For IDs I, J, if I yields J in one move of $\mathcal{M}$ then we write

$$I \vdash_{\mathcal{M}} J.$$

If ID I yields ID J in 0 or more steps then we write $I \vdash^*_{\mathcal{M}} J$.(If $\mathcal{M}$ is clear from the context we omit $\mathcal{M}$.) Thus $I \vdash^* J$ if $I = J$ or there is a sequence of ID's $I_0, I_1, \ldots, I_n$ such that $I_0 = I, I_n = J$ and for each $k < n, I_k \vdash I_{k+1}$.

A string $w \in \Sigma^*$ is accepted by $\mathcal{M}$, if starting from the initial configuration, in finitely many moves, $\mathcal{M}$ enters an accepting state.
The **language accepted by** $\mathcal{M}$ is

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma^* : q_0 w \vdash^* I, \text{ for some accepting ID I}\}.$$

Thus $\mathcal{L}(\mathcal{M})$ is the set of all strings over Σ accepted by $\mathcal{M}$. Such languages are called **Turing-recognizable** or **recursively enumerable (re).**

Remark 1. We may w.l.g. assume that a TM $\mathcal{M}$ halts whenever it enters an accepting state. This may be done as follows. Add a new "halting state" q_h. Modify the transition function such that from any accepting state $\mathcal{M}$ enters the state q_h. Thus whenever $\mathcal{M}$ enters an accepting state, it then immediately enters q_h and halts i.e. it has no next move.

Definition 2. *A language is said to be* ***decidable*** *or* ***recursive*** *if it is accepted by a Turing machine that* ***halts on every input****.*

Simple examples:

Example 1. We shall construct a TM $\mathcal{M}$ that accepts all palindromes over the alphabet $\{0, 1\}$. We first given an informal description of $\mathcal{M}$. Given an input, $\mathcal{M}$ reads the first letter of the input and "marks" it. If it is a 0 (respectively 1) it marks it with an X (respectively Y). It then moves right skipping over the 0's and 1's until it finds the *last* unmarked letter. If it matches with the letter already marked, then it replaces it with an X or a Y as the case may be. It then moves left skipping over the 0's and 1's until it finds the first unmarked letter of the input. $\mathcal{M}$ then repeats the process. If after marking the last unmarked letter, $\mathcal{M}$ does not find any unmarked letter, then the input is an even palindrome and $\mathcal{M}$ accepts. If after marking the first unmarked symbol, $\mathcal{M}$ does not find any unmarked letter, then the input is an odd palindrome and $\mathcal{M}$ accepts.

Exercise 1. Give a formal definition of the transition function of $\mathcal{M}$.
Hence describe, formally, an accepting computation on input 10101

Remark 2. The TM constructed above halts on every inputs and hence the set of palindromes over $\{0, 1\}$ is recursive.

Exercise 2. Construct a Turing machine that accepts

$$\{ww : w \in \{0, 1\}^*\}.$$

1.1 Variants of TMs:

1. Multi-tracks single-tape TM.
2. Single-tape one-way infinite.
3. Multi-tape TM

1. **Multi-track TM:** In a k-track TM $\mathcal{M}$, at any given instant, the tape-head would be scanning the contents of all the cells in the tracks. Thus, $\mathcal{M}$ would be reading a k-tuple of symbols . Depending on the k-tuple being read, and the state of the finite control, $\mathcal{M}$ would print a k-tuple of symbols–one for each track–move the tape-head one position to the right or left and enters a, possibly, new state. The input string is placed onto the first track with all other cells being blank. Thus if $a \in \Sigma$ is identified with the k-tuple $(a, B, \ldots, B)$ and the blank is identified with $(B, \ldots, B)$, then a muti-track TM is simply a single-tape TM whose tape alphabet is Γ^k with blank being identified with $(B, \ldots, B)$.
2. Exercise: Show that a two-way infinite tape TM can be simulated by a one-way infinite single tape TM.
3. **Multi-Tape TM:** In a $k-$tape TM $\mathcal{M}$, there are k tapes and for each tape there is a tape-head scanning the content of a cell at each instant. Depending on the letters being scanned by the tape-heads and the state of the finite-control, in one move $\mathcal{M}$

1. prints a letter in each of the cell being scanned
2. moves the tape-heads–independent of each other– one cell to the right or left, and
3. enters a possibly new state.

The input string is placed on the first tape with the corresponding tape-head scanning the first cell and all other cells are blank. The TM $\mathcal{M}$ starts in the initial state q_0 and if after finitely many moves, $\mathcal{M}$ enters an accepting state, the input string is accepted by $\mathcal{M}$. The language accepted by $\mathcal{M}$ is

$$\mathcal{L}(\mathcal{M}) = \{w \in \Sigma \mid \mathcal{M} \text{ accepts w}\}.$$

Thus the transition-function of a k-tape TM is a function of the form

$$\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k.$$

Theorem 1. *Every multi-tape TM $\mathcal{M}$ can be simulated by a single tape TM $\hat{\mathcal{M}}$.*
Consequently, $\mathcal{L}$ is re iff it is accepted by a multi-tape TM.

Proof. Let $\mathcal{M}$ be a k-tape TM. We shall construct a single-tape TM $\hat{\mathcal{M}}$ with $2k$ tracks–2 tracks for each tape– to simulate $\mathcal{M}$. The upper track contains the contents of the corresponding tape, while the lower track contains a single marker that indicates the position of the corresponding tape-head. To simulate a move of $\mathcal{M}$, $\hat{\mathcal{M}}$ starts from the cell containing the leftmost marker in the state of $\mathcal{M}$.

$\hat{\mathcal{M}}$ then makes a sweep from left to right visiting all the cells containing a marker and also noting the letters above the markers. A part of the finite control counts the number of markers visited as well as the letters being "read" by the markers. When $\hat{\mathcal{M}}$ has visited all the cells containing a marker, it has enough information to simulate a move of $\mathcal{M}$. $\hat{\mathcal{M}}$ then makes a leftward journey updating the letters above the markers and also moving the marker either to the left or to the right in order to simulate the move of $\mathcal{M}$. $\hat{\mathcal{M}}$ then enters the state to which $\mathcal{M}$ enters. $\hat{\mathcal{M}}$ accepts whenever $\mathcal{M}$ accepts.□

Exercise 3. Show that to simulate n moves of $\mathcal{M}$, $\hat{\mathcal{M}}$ requires $O(n^2)$ moves.

Non-deterministic TM: In a non-deterministic TM $\mathcal{M}$, at each instant, the TM –depending on the letter being scanned and the state of the finite control–has several choices for the next move. Thus the transition function is a function of the form

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\}).$$

The equation

$$\delta(q, a) = \{(p_1, a_1, D_1), \ldots, (p_k, a_k, D_k)\}$$

means that the TM in state q, reading the letter a, has the choice of printing a_i in place of a, enter the state p_i and move the tape-head in the direction D_i, where i is one of $1, 2, \ldots, k$.

A string w is accepted by $\mathcal{M}$ if, starting from the initial configuration, $\mathcal{M}$ has a choice of moves that will lead to an accepting configuration.

We shall now show that a non-deterministic TM is no more powerful that a deterministic one.

Theorem 2. *Every non-deterministic Turing machine $\mathcal{M}$ can be simulated by a deterministic Turing machine $\hat{\mathcal{M}}$.*

Consequently, a language $\mathcal{L}$ is re iff it is accepted by non-deterministic Turing machine.

Proof. Let us view $\mathcal{M}$'s computation on w as a tree where each node is labelled by a configuration, while the children of a node, labelled by I, are labelled by the all the possible configurations that $\mathcal{M}$ can enter in one move from the configuration I. The root is labelled by the initial configuration. The TM $\hat{\mathcal{M}}$ explores the tree by using the breath-first search. As soon as $\hat{\mathcal{M}}$ finds an accepting configuration, it halts and accepts.

Thus, $\hat{\mathcal{M}}$ consists of three tapes. The first tape is the input tape, while the second tape is the simulation or computation tape. The third tape is an address tape. Suppose each node of the tree has at most k children. Then each node is assigned an address which is a string over the alphabet $\{1, 2, \ldots, k\}$. For instance 231 is assigned to the node we arrive at by starting at the root, then going to its 2nd child, then going to that node's 3rd child and finally going to that node's 1st child. The TM $\hat{\mathcal{M}}$ behaves as follows. Initially, the first tape contains the input w, while the other tapes are blank.

1. Copy the input w onto the second tape.
2. Simulate $\mathcal{M}$ on w as dictated by the address on the 3rd tape, aborting if the address is invalid. Accept the input string if the configuration on the node visited is accepting.
3. Replace the string on the 3rd tape with the lexicographically next string and go to step 2.

Clearly, $\hat{\mathcal{M}}$ simulates $\mathcal{M}$. □

Exercise 4. Show that to simulate n moves of $\mathcal{M}$, $\hat{\mathcal{M}}$ requires about $O(2^n)$ steps.

1.2 Properties of re and recursive languages.

Theorem 3. *A language $\mathcal{L} \subseteq \Sigma^*$ is recursive iff $\mathcal{L}$ and $\mathcal{L}^C$ are both re.*

Proof. Suppose $\mathcal{L}$ is recursive. Fix a TM $\mathcal{M}$ that accepts $\mathcal{L}$ and halts on all inputs. Clearly $\mathcal{L}$ is re. Let $\mathcal{M}'$ be the TM obtained from $\mathcal{M}$ by interchanging the accepting and rejecting states.Then clearly, $\mathcal{M}'$ accepts $\mathcal{L}^C$.

Conversely, suppose both $\mathcal{L}$ and $\mathcal{L}^C$ are re. Fix single-tape TMs $\mathcal{M}_1, \mathcal{M}_2$ accepting $\mathcal{L}, \mathcal{L}^C$ respectively. W.l.g. assume that both $\mathcal{M}_1$ and $\mathcal{M}_2$ halt whenever they enter an accepting state i.e. both $\mathcal{M}_1, \mathcal{M}_2$ halt on acceptance. Construct a two-tape TM $\hat{\mathcal{M}}$ as follows.

On input w, $\hat{\mathcal{M}}$ copies w onto the second tape and then runs $\mathcal{M}_1$ on tape-1 and $\mathcal{M}_2$ on tape-2. $\hat{\mathcal{M}}$ accepts whenever $\mathcal{M}_1$ accepts and halts when one of them halts. Clearly, $\hat{\mathcal{M}}$ accepts $\mathcal{L}$ and for any input w either $w \in \mathcal{L}$ or $w \in \mathcal{L}^C$ and hence one of $\mathcal{M}_1, \mathcal{M}_2$ halts on input w. So $\hat{\mathcal{M}}$ halts on any input. Hence, $\mathcal{L}$ is recursive. □

Theorem 4. *Recursive/re languages are closed under finite $\bigcup$ and $\bigcap$.*
Recursive languages are closed under complementation.

Proof. We shall prove closure under finite $\bigcup$ for re. The remaining proofs are similar.

Let $\mathcal{L}_1, \mathcal{L}_2$ be re languages. Fix single-tape TMs $\mathcal{M}_1, \mathcal{M}_2$ accepting $\mathcal{L}_1, \mathcal{L}_2$ respectively. We shall construct a two-tape TM $\hat{\mathcal{M}}$ as follows.

On input w, $\hat{\mathcal{M}}$ copies w onto the second tape and then runs $\mathcal{M}_1$ on tape-1 and $\mathcal{M}_2$ on tape-2. $\hat{\mathcal{M}}$ accepts whenever one of $\mathcal{M}_1, \mathcal{M}_2$ accepts. Clearly, $\hat{\mathcal{M}}$ accepts $\mathcal{L}_1 \bigcup \mathcal{L}_2$. □

1.3 Undecidability

Encoding TMs: Let $\mathcal{M} =< \{0,1\}, \Gamma, Q, q_1, \delta, B, F >$ be a Turing machine over the alphabet $\{0,1\}$. Let

$$Q = \{q_1, q_2, \ldots, q_n\},$$

where q_1 is the initial state and q_2 is the only accepting state. Let

$$\Gamma = \{X_1, X_2, X_3, \ldots, X_k\},$$

for some integer k, where $X_1 = 0, X_2 = 1$ and $X_3 = B$. Let $D_0 = L, D_1 = R$. Then each move or transition of a TM $\mathcal{M}$ is given by an equation of the form

$$\delta(q_i, X_j) = (q_k, X_l, D_m), \tag{1}$$

where $1 \leq i, k \leq n, 1 \leq j, l \leq k$ and $m = 1, 2$. This can be encoded by the binary string

$$0^i 10^j 10^k 10^l 10^m$$

The collection of these binary encodings completely describes δ and can be combined to give an encoding of the TM $\mathcal{M}$ as follows.
Let $C_1, C_2, \ldots, C_m$ be the encodings of all the transitions of $\mathcal{M}$. Then a code of the entire TM $\mathcal{M}$ is

$$C_1 11 C_2 11 \ldots 11 C_m$$

and is denoted by $< \mathcal{M} >$.

This also gives an enumeration of all TM's over the alphabet $\{0,1\}$ as follows.
Let $w_1, w_2, \ldots$ be the canonical enumeration of all binary strings. Thus $w_1 = 0, w_2 = 1, w_3 = 00, w_4 = 01, w_5 = 10$ and so on. Then the ith Turing machine, denoted by $\mathcal{M}_i$, is the Turing machine over $\{0,1\}$ whose code is w_i. If w_i is not a code of a TM, then $\mathcal{M}_i$ is a fixed TM $\mathcal{M}^*$ such that $\mathcal{L}(\mathcal{M}^*) = \phi$. Note that $< \mathcal{M}_i >= w_i$, if w_i is a code of a TM.

Exercise 5. Find an algorithm $\mathcal{A}$ which when given an input a binary string w outputs an integer i such tha $w = w_i$.

What is the complexity of your algorithm?

Define the *diagonalization language* L_d by

$$\mathcal{L}_d = \{w_i : \mathcal{M}_i \text{ does not accept } \mathrm{w_i}\}.$$

Theorem 5. *The language L_d is not recursively enumerable*

Proof. Suppose L_d is recursively enumerable. Then there is a TM, say $\mathcal{M}_{i^*}$, such that

$$\mathcal{L}(\mathcal{M}_{i^*}) = L_d.$$

Now consider the string w_{i^*}. If $w_{i^*} \in L_d$, then by our assumption, $\mathcal{M}_{i^*}$ accepts w_{i^*}. This implies that $w_{i^*} \notin L_d$, a contradiction. On the other hand, if $w_{i^*} \notin L_d$, then $\mathcal{M}_{i^*}$ does not accepts w_{i^*}. By definition of L_d, this means that $w_{i^*} \in L_d$, again a contradiction. Thus L_d cannot be re. □

Universal language: Let

$$\mathcal{L}_U = \{<\mathcal{M}> 111w : \mathcal{M} \text{ accepts w}\}.$$

We shall show that $\mathcal{L}_U$ is re but **not** recursive. The language $\mathcal{L}_U$ is called a universal language and the TM accepting $\mathcal{L}_U$ is called a universal TM.

Theorem 6. *The language $\mathcal{L}_U$ is recursively enumerable.*

Proof. We shall construct a 3-tape TM $\mathcal{M}_U$ that accepts $\mathcal{L}_U$. $\mathcal{M}_U$ works as follows.

1. $\mathcal{M}_U$ first checks that the input is of the form $<\mathcal{M}> 111w$. If not, then $\mathcal{M}_U$ rejects the input string.
2. $\mathcal{M}_U$ then copies w onto the second tape. Note that w is the string that follows the first block of 111.
3. The third tape contains a string of 0's with 0^i representing the state q_i. Initially, the third tape consists of 0 to represent the initial state q_1.
4. To simulate a move of $\mathcal{M}$, $\mathcal{M}_U$ searches within $<\mathcal{M}>$ a substring of the form $0^i10^j10^k10^l10^m$, where 0^i is the string on the 3rd tape and X_j is the symbol of $\mathcal{M}$ at the position on tape 2 scanned by $\mathcal{M}_U$. This represents the transition that $\mathcal{M}$ would next make. Thus $\mathcal{M}_U$ behaves as follows.
   (a) Change the contents of tape 3 to 0^k i.e. simulate the state change of $\mathcal{M}$
   (b) Replace X_j by X_l i.e. change the tape symbol of $\mathcal{M}$,
   (c) Shift the tape head on tape 2 one cell to the left if $m = 1$ or one cell to the right if $m = 2$. Thus $\mathcal{M}_U$ simulates the move of $\mathcal{M}$ to the left or to the right.
5. If no such substring is found, then $\mathcal{M}$ has no next move and hence halts in the simulated configuration. $\mathcal{M}_U$ does likewise.
6. If $\mathcal{M}$ enters the accepting state, then the contents of tape 3 is 00 and so $\mathcal{M}_U$ accepts.

In this way, $\mathcal{M}_U$ simulates $\mathcal{M}$ on w and accepts $<\mathcal{M}> 111w$ iff $\mathcal{M}$ accepts w. □

Theorem 7. *The universal language $\mathcal{L}_U$ is not recursive.*
Consequently, the class of recursive languages is strictly contained in the class of re languages.

Proof. Suppose $\mathcal{L}_U$ is recursive. Then $\mathcal{L}_U^C$ is re. Hence there is a TM $\mathcal{M}$ accepting $\mathcal{L}_U^C$. Using $\mathcal{M}$ we shall construct a TM $\mathcal{M}'$ that accepts $\mathcal{L}_d$. Given input w_i, $\mathcal{M}'$ first checks if w_i is a code of a TM. If it is not a code, then $\mathcal{M}_i = \mathcal{M}^*$ and hence $\mathcal{M}_i$ doe not accept w_i and so $\mathcal{M}'$ accepts w_i. Suppose w_i is a code, then it is a code of the ith TM $\mathcal{M}_i$. The TM $\mathcal{M}'$ then runs $\mathcal{M}$ on w_i111w_i. If $\mathcal{M}$ accepts, then $\mathcal{M}'$ accepts w_i. Note that $\mathcal{M}$ accepts w_i111w_i iff $<\mathcal{M}_i> 111w_i \notin \mathcal{L}_U$ iff $\mathcal{M}_i$ does not accepts w_i. Thus $\mathcal{M}_i$ accepts $\mathcal{L}_d$. This contradicts the fact that $\mathcal{L}_d$ is not re. Thus $\mathcal{L}_U$ can not be recursive.□

1.4 Intractable Problems

Definition 3. *A function*

$$f : \Sigma_1^* \to \Sigma_2^*$$

*is said to be **polynomial-time computable** if there is a polynomial $p(x)$ and a TM $\mathcal{M}$ with an **output** tape such that for every input w of length n, in at most $p(n)$ moves of $\mathcal{M}$, $f(w)$ is obtained on the output tape of $\mathcal{M}$.*

Definition 4. *Let $\mathcal{L}_1, \mathcal{L}_2$ be two languages.*

*$\mathcal{L}_1$ is **polynomial-time reducible** to $\mathcal{L}_2$ and we write $\mathcal{L}_1 \leq_p \mathcal{L}_2$ if there is a polynomial-time computable function f such that*

$$w \in \mathcal{L}_1 \leftrightarrow f(w) \in \mathcal{L}_2.$$

Intuition: $\mathcal{L}_1$ is not more complex than $\mathcal{L}_2$.

Proposition 1. *$\leq_p$ is reflexive and transitive.*

Proof. Clearly, $\leq_p$ is reflexive. For transitivity, let $\mathcal{L}_1 \leq_p \mathcal{L}_2$ and $\mathcal{L}_2 \leq_p \mathcal{L}_3$. Fix two polynomial-time computable functions f, g such that

$$w \in \mathcal{L}_1 \leftrightarrow f(w) \in \mathcal{L}_2,$$

$$x \in \mathcal{L}_2 \leftrightarrow g(x) \in \mathcal{L}_3.$$

Set $h = g \circ f$. Then

$$w \in \mathcal{L}_1 \leftrightarrow f(w) \in \mathcal{L}_2 \leftrightarrow g(f(w)) \in \mathcal{L}_3,$$

i.e.

$$w \in \mathcal{L}_1 \leftrightarrow h(w) \in \mathcal{L}_3.$$

We claim that h is a polynomial-time computable function.

Let $p(n), q(n)$ be the polynomial bounds for the functions f, g respectively. Let $\mathcal{M}_1, \mathcal{M}_2$ be the TMs computing f, g respectively. We shall construct a TM $\mathcal{M}$ by combining $\mathcal{M}_1, \mathcal{M}_2$ as follows. On input w of length n, $\mathcal{M}$ first runs $\mathcal{M}_1$ on w and obtains $f(w)$ on the output tape in at most $p(n)$ moves. Since in one move, $\mathcal{M}_1$ can print only one symbol, the length of $f(w)$ is at most $p(n)$.$\mathcal{M}$ now runs $\mathcal{M}_2$ on $f(w)$ and obtains $g(f(w)) = h(w)$ on the output tape. Note that $\mathcal{M}_2$ takes at most $q(p(n))$ steps to compute $g(f(w)) = h(w)$. Thus the total time taken by $\mathcal{M}$ to compute $h(w)$ is at most $p(n) + q(p(n))$, which is a polynomial. Thus h is polynomial-time computable function.

Remark 3. The proof shows that if f and g are polynomial-time computable functions then so is $h = g \circ f$.

Definition 5. *A TM $\mathcal{M}$ is said to be $T(n)$-time bounded if for every input w of length n accepted by $\mathcal{M}$, it makes at most $T(n)$ moves to accept w. It is said to be poly-time bounded if it is $p(n)$- time bounded for some polynomial $p(.)$.*

The Class $\mathcal{P}$, $\mathcal{NP}$:
The language $\mathcal{L}$ is in the class $\mathcal{P}$ (resp. $\mathcal{NP}$) if $\mathcal{L}$ is accepted by a poly-time bounded **deterministic** (resp **non-deterministic**) TM $\mathcal{M}$.

Definition 6. *A language $\mathcal{L}$ is said to be $\mathcal{NP}$-complete if*

1. *$\mathcal{L} \in \mathcal{NP}$.*
2. *For every $\mathcal{L}'$ in $\mathcal{NP}, \mathcal{L}' \leq_p \mathcal{L}$.*

If only condition (2) holds then it is said to be $\mathcal{NP}$-hard.

Theorem 8. *If $\mathcal{L} \in \mathcal{P}$ and $\hat{\mathcal{L}} \leq_p \mathcal{L}$ then $\hat{\mathcal{L}}$ is also in $\mathcal{P}$.*

Proof. Since $\hat{\mathcal{L}} \leq_p \mathcal{L}$, there is $p(n)$-time computable function f such that

$$w \in \hat{\mathcal{L}} \leftrightarrow f(w) \in \mathcal{L}.$$

Let $\mathcal{M}_1$ be a $p(n)$-time bounded TM computing f and let $\mathcal{M}_2$ be a $q(n)$-time bounded TM accepting $\mathcal{L}$. We shall construct a poly-time bounded TM $\hat{\mathcal{L}}$ that accepts $\hat{\mathcal{L}}$ as follows.

On input w of length $n, \hat{\mathcal{M}}$ first runs $\mathcal{M}$ on w to obtain $f(w)$ in at most $p(n)$ steps. $\hat{\mathcal{M}}$ then runs $\mathcal{M}_2$ on $fw)$ and accepts w iff $\mathcal{M}_2$ accepts $f(w)$. Clearly, $\hat{\mathcal{M}}$ accepts $\hat{\mathcal{M}}$. Now observe that $|f(w)|$ is at most $p(n)$ and hence $\mathcal{M}_2$ on $f(w)$ takes at most $q(p(n))$ steps on accetance. Thus the total time taken by $\hat{\mathcal{M}}$ in accepting w is at most $p(n) + q(p(n)) \overset{\text{def}}{=} r(n)$. Hence, $\hat{\mathcal{M}}$ is a poly-time bounded TM accepting $\hat{\mathcal{L}}$ and so $\hat{\mathcal{L}} \in \mathcal{P}$. □

Theorem 9. *If $\mathcal{L}$ is $\mathcal{NP}$-complete and $\mathcal{L} \in \mathcal{P}$, then $\mathcal{P} = \mathcal{NP}$.*

Proof. Suffices to show that $\mathcal{NP} \subseteq \mathcal{P}$. So fix $\mathcal{L}' \in \mathcal{NP}$. Since $\mathcal{L}$ is $\mathcal{NP}$-complete, $\mathcal{L}' \leq_p \mathcal{L}$. Since $\mathcal{L}$ is in $\mathcal{P}$, by Theorem 8 $\mathcal{L}'$ is also in $\mathcal{P}$. This completes the proof. □

Remark 4. This gives us a technique of proving that a given language $\mathcal{L}$ is $\mathcal{NP}$-complete.

The first $\mathcal{NP}$-complete problem:

Definition 7. *Let ϕ be a Boolean formula in Conjunctive Normal Form (CNF). ϕ is said to be **satisfiable** if there is a truth assignment to the atoms in ϕ that makes ϕ TRUE. Let SAT be the set of all satisfiable formulas in CNF.*

Encode ϕ over some alphabet. (Any reasonable encoding is good enough.) Denote it by $<\phi>$. The language associated with SAT is the language

$$\mathcal{L}_{SAT} = \{<\phi>: \phi \text{ is in CNF and } \phi \text{ is satisfiable}\}.$$

In what follows, for simplicity, we shall work work with SAT rather that $\mathcal{L}_{SAT}$.

Cook-Levin Theorem:

Theorem 10. *SAT is $\mathcal{NP}$-complete.*

Consequently, $\mathcal{P} = \mathcal{NP}$ iff SAT $\in \mathcal{P}$.

Proof. We first show that $SAT \in \mathcal{NP}$. So fix a Boolean formula ϕ in CNF. Consider the following algorithm that can be easily simulated by a non-deterministic TM.

1. Non-deterministically choose an assignment of truth values to the atoms in ϕ.
2. Check in polynomial time if ϕ is TRUE under this assignment.

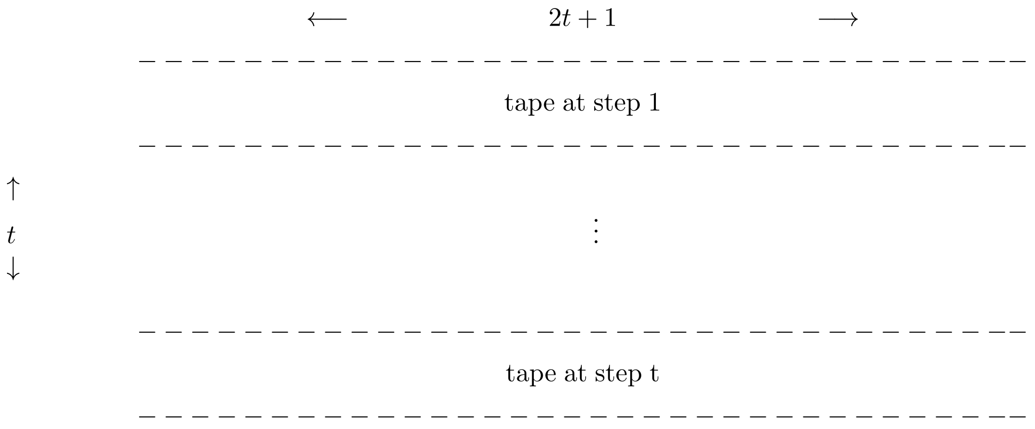
Note that if $\phi \in SAT$, then for some choice in step 1, ϕ will be TRUE. Thus SAT is in $\mathcal{NP}$.

We now show that SAT is $\mathcal{NP}$-hard. So fix a language $\mathcal{L} \in \mathcal{NP}$. Let $\mathcal{M}$ be a $p(n)$-bounded non-deterministic TM that accepts $\mathcal{L}$. Without loss of generality assume that the polynomial $p(n) \geq n$ for all n. Our aim is to find a polynomial-time computable function that transform a given string u into a CNF formula δ_u such that

$$u \text{ is accepted by } \mathcal{M} \text{ iff } \delta_{\text{u}} \text{ is satisfiable.}$$

For a given string u, let $t = p(|u|)$.

Observe that if $\mathcal{M}$ accepts the input u then it does so in at most t steps. Thus to determine if $\mathcal{M}$ accepts u we need to run it on u for at most t steps and then check if the configuration is an accepting configuration. Since in one move, $\mathcal{M}$ moves the tape head one cell to the left or right of the current cell scanned by it, in t moves, the tape head is at most t cell to the left or t cells to the right of the current position. Since $t \geq |u|$ it suffices to consider $2t+1$ cells of the tape for the computation by $\mathcal{M}$. Since we are considering t steps of the computation, the information of the entire computation of $\mathcal{M}$ can be revealed in the following $t \times (2t+1)$ array.



Without loss of generality we assume that if $\mathcal{M}$ accepts u in $< t$ steps then the last configuration is repeated so that $\mathcal{M}$ accepts u in exactly t steps. Thus the first row of the array corresponds to the initial configuration and is of the form

$$s_0^t u s_0^{t+1-|u|},$$

where $s_0 = B$ with $\mathcal{M}$ in state q_1 is scanning the first letter of u. The last row of the array corresponds to an accepting configuration.

Let the set of states of $\mathcal{M}$ be $Q = \{q_1, q_2, \ldots, q_n\}$, where q_1 is the initial state and q_n is the only accepting state. The tape alphabet is $\Gamma = \{s_0, s_1, \ldots, s_r\}$, where $s_0 = B$. We will define a CNF formula δ_u which is satisfiable iff $\mathcal{M}$ accepts u. Our set of atoms is

$$\mathcal{A} = \{\rho_{h,j,k}, \sigma_{i,j,k} : 1 \leq h \leq n, 1 \leq i \leq r, 1 \leq j \leq 2t+1, 1 \leq k \leq t\}.$$

We first assume that $\mathcal{M}$ accepts u, so that there is an accepting computation of $\mathcal{M}$ on u. We assume that the above $t \times (2t+1)$ array has been constructed accordingly. We shall construct a CNF formula δ_u that is satisfiable under the following assignment ν.

$$\nu(\rho_{h,j,k}) = \begin{cases} 1 \text{ if } \ \mathcal{M} \ \textit{in state } q_h \textit{ is scanning the } j\textit{th position at step } k \textit{ of the computation} \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

$$\nu(s_{i,j,k}) = \begin{cases} 1 \text{ if } \ \textit{the tape symbol } s_i \textit{ is in } j\textit{th position of the } k\textit{th row of the array} \\ 0 \text{ otherwise} \end{cases} \tag{3}$$

To describe δ_u we need the following

$$\nabla\{x_e : 1 \leq e \leq l\} \overset{\text{def}}{=} \bigwedge_{1 \leq e < f \leq l} (\neg x_e \vee \neg x_f) \wedge (\bigvee_{1 \leq e \leq l} x_e),$$

where $\{x_e : 1 \leq e \leq l\}$ is a set of atoms/formulas. It is not hard to check that $\nabla\{x_e : 1 \leq e \leq l\}$ is TRUE under an asignment of truth values iff exactly one of $x_1, x_2, \ldots, x_l$ is TRUE under that assignment. Also the length of the formula is $O(l^2)$.

The formula δ_u is the conjunction of the following 5 CNF formulas that describes the computation of $\mathcal{M}$ on u and is TRUE under the above assignment. Let $u = s_{u_1} \ldots . s_{u_z}$ where $z = |u|$.

1. " The first row of the array corresponds to the initial configuration with $\mathcal{M}$ in state q_1 scanning the first letter of u."
   This is expressed by

   $$\bigwedge_{1 \leq j \leq t} \sigma_{0,j,1} \wedge \bigwedge_{1 \leq j \leq z} \sigma_{u_j,t+j,1} \wedge \bigwedge_{1 \leq j \leq t+1-z} \sigma_{0,t+z+j,1} \wedge \rho_{1,z+1,1}.$$

   Clearly, this is of length $O(t)$.

2. ”Each entry of the array contains exactly one tape symbol.”
   This is expressed by
   $$\bigwedge_{1\le j\le 2t+1}\bigwedge_{1\le k\le t}\nabla\{\sigma_{i,j,k}:1\le i\le r\}.$$
   The length of the formula is $O(t^2)$.
3. ”At each step of the computation $\mathcal{M}$ is in a unique state scanning a unique cell.”
   This is expressed by
   $$\bigwedge_{1\le k\le t}\nabla\{\rho_{h,j,k}:1\le h\le n, 1\le j\le 2t+1\}.$$
   This expression is of length $O(t^3)$.
4. ”Each configuration of the computation, after the first, is either identical to the previous configutration or follows from it by the application of one of the transition rules of $\mathcal{M}$.”
   Let the transition rules of $\mathcal{M}$ be as follows.
   $$\{\delta(q_{i_a},s_{j_a})=(q_{k_a},s_{l_a},R):1\le a\le\bar{a}\},\tag{4}$$
   $$\{\delta(q_{i_b},s_{j_b})=(q_{k_b},s_{l_b},L):1\le b\le\bar{b}\}.\tag{5}$$
   The expression will be of the form
   $$\bigwedge_{1\le j\le 2t+1}\bigwedge_{1\le k\le t}(NOHEAD(j,k)\vee IDENT(j,k)\vee A(j,k)\vee B(j,k)),\tag{6}$$
   where each of the disjunct will be explained below. Each disjunct will be of constant length so that the expression will be of length $O(t^2)$.
   we define
   $$NOHEAD(j,k):=\bigvee_{1\le i\le r}(\sigma_{i,j,k}\wedge\sigma_{i,j,k+1})\wedge\bigwedge_{1\le h\le n}\neg\rho_{h,j,k}.$$
   Thus $\nu(NOHEAD(j,k))=1$ iff $\mathcal{M}$ is not scanning the jth cell at step k of the computaion.
   Next we define
   $$IDENT(j,k):=\bigvee_{1\le h\le n}\bigvee_{1\le i\le r}(\rho_{h,j,k}\wedge\sigma_{i,j,k}\wedge\rho_{h,j,k+1}\wedge\sigma_{i,j,k+1}),$$
   so that $\nu(IDENT(j,k))=1$ iff $\mathcal{M}$ is scanning the jth cell at both the kth and k_1st step of the computation, both the state and the symbol are the same in both the configurations.
   Next we define for $j\ne 2t+1$
   $$A(j,k):=\bigvee_{1\le a\le\bar{a}}(\rho_{i_a,j,k}\wedge\sigma_{j_a,j,k}\wedge\rho_{k_a,j+1,k+1}\wedge\sigma_{l_a,j,k+1}),$$
   so that $\nu(A(j,k))=1$ iff the $k+1$st step is obtained from the kth step by application of one of the rules of (4).
   Similarly, we define for $j\ne 1$
   $$B(j,k):=\bigvee_{1\le b\le\bar{b}}(\rho_{i_b,j,k}\wedge\sigma_{j_b,j,k}\wedge\rho_{k_b,j-1,k+1}\wedge\sigma_{l_b,j,k+1}),$$
   so that $\nu(B(j,k))=1$ iff the $k+1$st step is obtained from the previous step by the application of one of the rules of (5).
   Finally, the expression within (,) in (6) is converted into CNF.

5. "The configuration at step t is an accepting configuration."
   This is expressed by
   $$\bigvee_{1\leq j\leq 2t+1} \rho_{n,j,t}.$$
   This is of length $O(t)$.

δ_u is the conjunction of the CNF formulas (1)-(5) above. Thus if $\mathcal{M}$ accepts u, the CNF formula δ_u that is TRUE under the assignment ν described in (2) and (3).

Conversely, suppose δ_u is TRUE under a certain assignment ν. We now intrepret ν as in equations (2),(3). Then we can construct a $t\times(2t+1)$ array as follows. The CNF fomula (2) says that each cell of the array is occupied by a unique tape symbol. Formula (1) says that the first row of the array corresponds to the initial configuration with $\mathcal{M}$ in state q_1 scanning the first letter of u. Formula (3) says that at each step of the computation, $\mathcal{M}$ is in a unique state scanning a unique cell. Formula (4) is satisfiable under the assignment means that each configuration after the first, is either identical to the previous configuration or follows from it by the application of one of the rules of $\mathcal{M}$. Formula (5) says that the configuration at step t is an accepting computation. Thus the $t\times(2t+1)$ array reconstructs an accepting computation of $\mathcal{M}$ on u. So $\mathcal{M}$ accepts u.

This completes the proof. □

1.5 Other $\mathcal{NP}$-complete problems.

Definition 8. *A Boolean formula is said to be 3CNF if it is a CNF formula in which each clause contains at most* 3 *literals. Define*

$$3SAT=\{\phi:\phi \text{ is a satisfiable 3CNF formula}\}.$$

The corresponding laguage is

$$\mathcal{L}_{3SAT}=\{<\phi>:\phi\in 3SAT\}.$$

We next prove

Theorem 11. *3SAT is $\mathcal{NP}$-complete.*

Proof. Since SAT is in $\mathcal{NP}$ it easily followws that 3SAT is in $\mathcal{NP}$. To show 3SAT is $\mathcal{NP}$ -hard we shall reduce SAT to 3SAT. Our aim is to find a polynomial-time computable function that transform a CNF formula ϕ to a 3CNF formula $\hat{\phi}$ such that

$$\phi \text{ is satisfiable iff } \hat{\phi} \text{ is satisfiable.}$$

So fix a CNF formula ϕ. Let

$$C:=/\alpha_1\alpha_2\ldots\alpha_k$$

be a clause where $k\geq 4$. It suffices to transform C to a 3CNF formula $\hat{C}$ such that C is satisfiable iff $\hat{C}$ is satisfiable. Let $\beta_1,\beta_2\ldots\beta_{k-3}$ be atoms not in ϕ. We define

$$\hat{C}:=/\alpha_1\alpha_2\beta_1/\alpha_3\bar{\beta}_1\beta_2/\alpha_4\bar{\beta}_2\beta_3/\ldots/\alpha_{k-2}\bar{b}_{k-4}\beta_{k-3}/\alpha_{k-1}\alpha_k\bar{\beta}_{k-3}.$$

Claim: C is satisfiable iff $\hat{C}$ is satisfiable.

So assume $\nu(C)=1$ under some assignment ν to the variables in C. We shall extend ν to the variables $\beta_1,\ldots,b_{k-3}$ as follows.

Clearly $\nu(\alpha_l)=1$ for some $l, 1\leq l\leq k$. Set

$$\nu(\beta_i)=\begin{cases}1 \text{ if } & 1\leq i\leq l-2\\ 0 \text{ if } & l-1\leq i\leq k-3\end{cases}.$$

Note that the clauses $C_1, C_2, \ldots C_{l-2}$ contains $\beta_1, b_2, \ldots, \beta_{l-2}$ respectively and hence each of these clauses receives truth value 1. The $l-1$st clause contains α_l and hence is TRUE under the assignment ν. For $l \leq i \leq k-2, C_i$ contains the literal $\bar{b}_{i-1}$ and hence takes the truth value 1. Thus it follows that

$$\nu(\hat{C}) = 1.$$

Conversely suppose $\nu(\hat{C}) = 1$ under some assignment ν.

We claim that $\nu(\alpha_i) = 1$ for some $i, 1 \leq i \leq k$. If not, then $\nu(\alpha_i) = 0, \forall i$. Since the $\nu(C_1) = 1$, we must have $\nu(\beta_1) = 1$. By induction one can show that $\nu(b_i) = 1, \forall i, 1 \leq i \leq k-3$. This forces the last lause to be FALSE. This contradiction shows that $\nu(\alpha_i) = 1$ for some i. Thus $\nu(C) = 1$. This completes the proof. □

COMPLETE SUBGRAPH

We next consider the COMPLETE SUBGRAPH(CS) problem: Given a graph G and an integer k, does G have a complete subgraph of size k? We now show that CS is $\mathcal{NP}$-complete.

Note that a graph G can be encoded by its adjacency matrix or incidence matrix. Its code is denoted by $< G >$. So the corresponding language is

$$\mathcal{L}_{CS} = \{< G > \#0^k : G \text{ has a complete subgraph of size k}\}.$$

However, for simplicity we work with a graph G instead of its code.

Theorem 12. *COMPLETE SUBGRAPH is $\mathcal{NP}$-complete.*

Proof. It is easy to check that CS is in $\mathcal{NP}$. To show that CS is $\mathcal{NP}$-hard, we shall reduce SAT to CS. So fix a CNF formula ϕ. We shall construct a polynomial time computable function that transforms ϕ into a graph G_ϕ and an integer k such that

$$\phi \text{ is satisfiable iff } \mathrm{G}_\phi \text{ has a complete subgraph of size k.}$$

Let $\phi := /C_1/ \ldots /C_k$ where $C_1, \ldots, C_k$ are clauses. We construct $G_\phi = (V, E)$ as follows.

$$V = \{(\alpha, i) : \alpha \text{ is a literal in } \mathrm{C_i}\}.$$

$$E = \{\{(\alpha, i), (\beta, j)\} : i \neq j \text{ } and \text{ } \alpha \neq \neg\beta\}.$$

Claim: ϕ is satisfiable iff G_ϕ has a complete subgraph of size k.

First assume ϕ is satisfiable. Thus $\nu(\phi) = 1$ under some assignment ν. For each $i, 1 \leq i \leq k$, fix a literal α_i in clause C_i such that $\nu(\alpha_i) = 1$. It is easy to check that $(\alpha_1, 1), \ldots, (\alpha_k, k)$ forms a complete subgraph of size k in G_ϕ. Conversely, let $(\alpha_1, i_1), \ldots, (\alpha_k, i_k)$ be the vertices of a complete subgraph of G_ϕ of size k. Clearly, by definition, $i_1, \ldots, i_k$ must be distinct integers and hence is a permutation of $1, \ldots, k$. Moreoever, α_t is a literal in $C_{i_t}, 1 \leq t \leq k$. Define an assignment ν such that $\nu(\alpha_t) = 1, \forall t$. Clearly ν is a valid assignment and $\nu(\phi) = 1$. Thus the claim holds. Further, G_ϕ can be obtained from ϕ by a polynomial time computable function. □

VERTEX COVER

Given a graph $G = (V, E)$, let $\bar{G} = (V, \bar{E})$ denote its complement. For the Vertex Cover problem we first need the following definition.

Definition 9. *Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ is said to be a **vertex cover** of G if for every edge $\{u, v\} \in E$ either $u \in S$ or $v \in S$.*

The Vertex Cover problem is : given a graph G and an integer k, is there a vertex cover of size k? To show that the Vertex Cover problem is $\mathcal{NP}$-complete we need the following

Theorem 13. *Let $G = (V, E)$. Let $\bar{G} = (V, \bar{E})$ denote its complement i.e. $\{u, v\} \in \bar{E}$ iff $\{u, v\} \notin E$. Then $S \subseteq E$ is a set of vertices of a complete sungraph of G iff $V - S$ is a vetex cover of $\bar{G}$*

Proof. First suppose S forms a complete subgraph of G. Fix an edge $\{u, v\} \in \bar{E}$. Then $\{u, v\}$ is not an edge of G. Hence either $u \notin S$ or $v \notin S$ i,e, one of u, v is in $V - S$. Hence $V - S$ is a vertex cover of $\bar{G}$.

Conversely suppose $V - S$ forms a vertez cover of $\bar{G}$.
Claim: The vertices of S form a complete subgraph of G. So fix two vertices u, v in S. Since neither u nor v is in $V - S$, $\{u, v\}$ can not be an edge of $\bar{G}$. Hence $\{u, v\}$ is an edge of G. This completes the proof. □

Theorem 14. *VERTEX COVER is $\mathcal{NP}$-complete.*

Proof. Given a graph $G = (V, E)$, non-deterministically choose k vertices $v_1, \dots, v_k$ and check in time $O(k^2)$ if these vertices form a vertex cover of G. Thus $VERTEXCOVER$ is in $\mathcal{NP}$.

We now show that VERTEX COVER is $\mathcal{NP}$-hard by reducing COMPLETE SUBGRAPH to VERTEX COVER. Given a graph G with n vertices and an integer k, construct its complement $\bar{G}$. Then by Theorem 13, G has a complete subgraph of size k iff $\bar{G}$ has a vertex cover of size $n - k$. This is clearly a polynomial-time reduction. Thus VERTEX COVER is $\mathcal{NP}$-hard. □

SET COVER

The SET COVER problem: Given a family of sets $\varDelta = \{S_1, S_2, \dots, S_n\}$ and an integer k, determine whether there is a subfamily $\varGamma$ of size k, $\varGamma = \{S_{i_1}, S_{i_2}, \dots, S_{i_k}\}$ such that

$$\bigcup_{1 \le i \le n} S_i = \bigcup_{1 \le t \le k} S_{i_t}. \tag{7}$$

Theorem 15. *SET COVER is $\mathcal{NP}$-complete.*

Proof. Given the family $\varDelta$ non-deterministically choose sets $S_{i_1}, \dots, S_{i_k}$ from $\varDelta$ and check in polynomial time whether (7) holds.

We now reduce VERTEX COVER to SET COVER as follows.

Given a graph $G = (V, E)$ and an integer k, let $V = \{v_1, \dots, v_n\}$. For each $i, 1 \le i \le n$, construct

$$S_i = \{(v_i, v_j), (v_j, v_i) : \{v_i, v_j\} \in E\}.$$

It is not hard to check that $\{v_{i_1}, \dots, v_{i_k}\}$ is a vertex cover for G iff $\varGamma = \{S_{i_1}, \dots, S_{i_k}\}$ is a set cover for $\varDelta = \{S_1, \dots, S_n\}$. □

INDEPENDENT SET

Definition 10. *Let $G = (V, E)$ be a graph. A subset $I \subseteq V$ is said to be an independent set if for every pair of vertives $u, v \in I$, $\{u, v\}$ is not an edge of G.*

The INDEPENDENT SET problem is the following.

Given a graph $G = (V, E)$ and an integer k, determine whether G has an independent set of size k.

The following is easy to prove.

Theorem 16. *Let $G = (V, E)$ be a graph. Then $I \subseteq V$ is an independent set iff $V - I$ is a vertex cover for G.*

Using this one can show

Theorem 17. *INDEPENDENT SET is $\mathcal{NP}$-complete.*

GRAPH COLOURING:

Definition 11. *A graph G is said to be k-colourable if the vertices of G can be coloured with k colours such that if $\{u, v\}$ is an edge of G, then u and v receive different colours. Such a colouring is called a valid or an admissible colouring.*

GRAPH COLOURING problem: Given a graph G and an integer k, decide if G is k-colourable.

We now show that GRAPH COLOURING is $\mathcal{NP}$-complete.

Theorem 18. *The GRAPH COLOURING problem is $\mathcal{NP}$-complete.*

Proof. Given a graph G and an integer k, non-deterministically colour the vertices with colours $1, 2, \dots, k$ and check in polynomial time whether it is a valid colouring. Thus GRAPH COLOURING is in $\mathcal{NP}$. To complete the proof we shall reduce 3SAT to GRAPH COLOURING. So fix a 3CNF formula ϕ. Suppose

$$\phi := /C_1/ \dots /C_k,$$

where each C_i is a clause containing at most 3 literals. Without loss of generality assume $k \geq 4$.

Our aim is to construct a polynomial-time computable function that tranforms a given 3CNF formula ϕ into a graph G_ϕ and an integer k such that

$$\phi \ is \ satisfiable \leftrightarrow G_\phi \ is \ k \ colourable.$$

Let $x_1, x_2, \dots, x_n$ be the atoms present in ϕ. Construct a graph G_ϕ as follows. The set of vertices is

$$V = \{v_i : 1 \leq i \leq n\} \bigcup \{x_i, \bar{x}_i : 1 \leq i \leq n\} \bigcup \{c_i : 1 \leq i \leq k\}.$$

The set of edges is

$$E = \{\{v_i, v_j\} : 1 \leq i \neq j \leq n\} \bigcup \{\{x_i, \bar{x}_i\} : 1 \leq i \leq n\} \bigcup \{\{v_i, x_j\}, \{v_i, \bar{x}_j\} : 1 \leq i \neq j \leq n\}$$

$$\bigcup \{\{\alpha, c_j\} : \alpha \ is \ a \ literal \ and \ \alpha \notin C_j, 1 \leq j \leq k\}.$$

We shall show that ϕ is satisfiable iff G_ϕ is $n+1$-colourable.

So first assume that ϕ is TRUE under an assignment ν. Each vertex $v_i, 1 \leq i \leq n$, is coloured with colour i. Note that one of $x_i, \bar{x}_i, 1 \leq i \leq n$, can be coloured with colour i. To colour the other literal we need a new colour $n+1$. The literal that is TRUE under ν is given the colour i and the other literal is coloured with colour $n+1$. Now we show how to colour the vertices c_j's . Since $k \geq 4$, for each clause C_j there is a pair of literals $\{x_i, \bar{x}_i\}$ both not in C_j. Hence C_j can not be coloured with colour $n+1$. But C_j contains a TRUE literal α which receives some colour $i \neq n+1$. We colour C_j with that colour i. Clearly this is a valid $n+1$-colouring.

Conversely, assume that G_ϕ is $n+1$-colourable. Without loss of generality, assume that $v_i, 1 \leq i \leq n$ receives colour i. Now for each $i, 1 \leq i \leq n$, one of the literals $x_i, \bar{x}_i$ receives the colour i and the other the "false"colour $n+1$. Assign the truth value TRUE to the literal that receives colour i and FALSE to the other literal. Thus we have an assignment of truth values to the literals.

Claim: Under this assignment of truth values, ϕ is TRUE.

So fix a clause $C_j, 1 \leq j \leq k$. As above, C_j cannot be coloured with colour $n+1$. So C_j receives some colour $i \neq n+1$. Clearly, one of the literals $x_i, \bar{x}_i$ that receives the colour i must be in C_j. Hence one of the literals that is TRUE is in C_j. Hence C_j is TRUE under the above assignment. Thus ϕ is TRUE. Clearly, the function

$$\phi \rightarrow < G_\phi, n+1 >$$

is polynomial-time computable. This completes the proof. □

HAMILTONIAN PATH (HAMPATH):

Definition 12. *Let G be a directed graph. A directed path in G is called Hamiltonian if it goes through each vertex of G exactly once.*

HAMPATH Problem: Given a directed graph G and a pair of vertices (s, t), decide if there is a Hamiltonian path from s to t.

Remark 5. Analogously, one can define the HAMPATH problem for undirected graphs also.

Exercise 6. Show that HAMPATH is $\mathcal{NP}$-complete by reducing 3SAT to HAMPATH.