1 Non-deterministic Turing Machine

A **nondeterministic Turing machine** is a generalization of the standard TM for which every configuration may yield none, or one or **more than one** next configurations.

In contrast to the deterministic Turing machines, for which a computation is a sequence of configurations, a computation of a nondeterministic TM is a tree of configurations that can be reached from the start configuration.

In this tree, the children-nodes of a node are its next configurations. Thus, the configuration, whose state is either q_a, or q_r has no children-nodes.

A **nondeterministic Turing machine**, written $NDTM$, is a 7-tuple $M = (Q, \Sigma, \Gamma, \Delta, q_0, q_a, q_r)$, where all ingredients except for Δ are defined as before for the deterministic TM.

The transition function Δ is defined by

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

Given a pair (q, σ), the transition function of an NDTM may yield a set of triples $\{(p, \sigma', D)\}$; this set can be empty.

The mapping Δ is convenient to present as a set of 5-tuples:

$$\{(q, \sigma, p, \sigma', D)\}$$

A configuration of an NDTM may yield several (or none) configurations in one step.

An input to an NDTM is said to be **accepted** if **there exists at least one** node of the computation tree which is an accept-configuration. The path from the root to the accept-configuration is said to be

non-deterministically selected.

A non-deterministic Turing Machine is called

a decider

if all branches halt on all inputs.

If, for some input, all branches are rejected, then the input is rejected.

Proposition 1 *A language is Turing-recognizable (acceptable) iff some nondeterministic Turing machine recognizes (accepts) it.*

2 Examples of non-deterministic TMs

Example 1

Given a set $S = \{a_1, \ldots, a_n\}$ of integers, determine if there is a subset $T \subseteq S$ such that

$$\sum_{a_i \in T} a_i = \sum_{a_i \in S-T} a_i.$$

The **language** L corresponding to the problem.

Language:

$$L = \{a_1 a_2 \ldots a_n : \exists T \subseteq S, \text{ s.t. } \sum_{a_i \in T} a_i = \sum_{a_i \in S-T} a_i.\}$$

ND Turing Machine:

- Non-deterministically select $T \subseteq S$;
- Compute $P_1 = \Sigma_{a_i \in T}\, a_i$ and $P_2 = \Sigma_{a_i \in S-T}\, a_i$
- if $P_1 = P_2$, accept.

Example 2

Given a graph $G = (V, E)$ and an integer $k > 0$, determine if there is a subset $C \subseteq V$ such that

- $|C| \geq k$;
- every two vertices in C are adjacent (C *is a clique*).

Language L:

$$L = \{\langle G, k\rangle : G \text{ has a clique of size} \geq k.\}$$

ND Turing Machine:

- Non-deteministically select $C \subseteq V$;
- Check if $|C| \geq k$;
- Check if $\forall x, y \in C$, $xy \in E$;
- if all checks up, accept.

Example 3

Given a graph $G = (V, E)$ and an integer $k > 0$, determine if there is a path P in G such that

- the length of $P \geq k$;
- no two vertices in G are traced twice by P.

Language L

$$L = \{\langle G, k\rangle : G \text{ there is a path of length} \geq k.\}$$

ND Turing Machine:

- non-deterministically select a path of length $\geq k$;
- accept.

3 Computational Classes

Definition 1 $\mathcal{P}$ *is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine:*

$$\mathcal{P} = \bigcup_k TIME(n^k).$$

Examples of languages in $\mathcal{P}$

1. $PATH = \{\langle G, s, t\rangle$: G is a directed graph that has a path from s to $t\}$; Use Dijkstra's algorithm. $PATH \in TIME(n^2)$

2. $GCD = \{\langle a, b\rangle$: a and b are relatively prime integers.$\}$

```
procedure Euclid (a, b); /*recursive version */
      if     (b == 0)
             return a;
      else
             return Euclid (b, a mod b);
```

```
TM D:  if    (Euclid(a, b) == 1)
             ACCEPT;
       else  REJECT
```

Lemma 1

/* $F_0 = 0; F_1 = 1; F_2 = 1; \ldots, F_{k+2} = F_{k+1} + F_k.$ */

If $a > b \geq 0$, and `Euclid` *performs k recursive calls, then $a \geq F_{k+1}$ and $b \geq F_k$.*

Proof. By induction on k.

Base. If $k = 0$, then $b \geq 0 = F_0$. Since $a > b$, $a \geq 1 = F_1$.

Inductive Step.

Let the lemma be true for $k-1$ recursive calls of Euclid.

Since $k > 0$, we have $b > 0$, and Euclid(b, a `mod` b) is recursively called.

Euclid(b, a `mod` b) makes $k-1$ calls, so $b \geq F_{k-1+1}$ (the role of a is played by b). Furthermore,

$$b + (a \bmod b) = b + (a - \lfloor \frac{a}{b} \rfloor \times b) \leq a.$$

Since $a \geq b + (a \bmod b) \geq F_k + F_{k-1} = F_{k+1}$.

Corollary. $GCD \in TIME(n^3)$.

Proof. Follows from the formula

$$F_k = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^k - \left(\frac{1-\sqrt{5}}{2}\right)^k}{\sqrt{5}}. \qquad \blacksquare$$

4 Verifiable problems.

There are many solvable problems for which no polynomial decider was found. Many of such problems are

polynomially verifiable.

Example 4

`Set_Partition`*:*
the problem of deciding if a given set $S = \{a_i\}$ *of numbers can be partitioned into two subsets* R *and* T *so that* $\Sigma_{a_i \in R}\, a_i = \Sigma_{a_i \in T}\, a_i$.

`Set_Partition:` $\{\langle S = \{a_i\}\rangle : \exists R, T\ (R \cup T = S)$ *such that* $\Sigma_{a_i \in R}\, a_i = \Sigma_{a_i \in T}\, a_i\}$.

Example 5 `Hampath`*: the problem of deciding if a given graph has a Hamiltonian path connecting two given vertices of the graph.*

`Hampath:` $\{\langle G, s, t\rangle : G$ *is a directed graph with a Hamiltonian path from* s *to* $t\}$.

Example 6 `Composites`*: the problem of deciding if a given positive integer is composite:*

$$\texttt{Composites} = \{x : x = pq \textit{ for integers } p > 1 \textit{ and } q > 1.\}$$

Definition 2 *A* **verifier** *for a language A is an algorithm V, where*

$$A = \{w : V \text{ accepts } \langle w, c \rangle \text{ for some string } c.\}$$

A language is **polynomially verifiable** *if it has a verifier which runs in time polynomial in $|w|$.*

Definition 3 *$\mathcal{NP}$ is the class of languages that have polynomial verifiers.*

Proposition 2 *Every problem in $\mathcal{P}$ belongs to $\mathcal{NP}$ ($\mathcal{P} \subseteq \mathcal{NP}$.)*

Proposition 3 *$\mathcal{NP}$ is the class of languages that are accepted by a non-deterministic TM in a polynomial time.*

Proof.

$\Rightarrow$ Let $A \in \mathcal{NP}$ and let V be a polynomial verifier for A (*exists by the definition of* $\mathcal{NP}$). If V runs in $O(|w|^k)$ time for some $k > 0$, then polynomial time NTM N is constructed by

On input w
non-deterministically select a string c
of length $O(n^k)$
run V on $\langle w, c \rangle$;
if V accepts, $ACCEPT$; else $REJECT$

$\Leftarrow$ If A is decided by a poly-time NTM N, we construct a verifier V by

On input $\langle w, c \rangle$, where w and c are strings
simulate N on w using c as a description of
the proper branch of the computation tree;
if this branch of N's computation accepts,
$ACCEPT$; else $REJECT$.

Theorem 1 *(Cook (1971) and Levin (1973))*

Class $\mathcal{NP}$ *has a problem* U *such that* $\mathcal{P} = \mathcal{NP}$ *iff* $U \in \mathcal{P}$.

Definition 4 *A graph $G(V, E)$ is called* **connected**, *if for any two distinct vertices u and v, there is a path connecting u with v.*

A **connected component** *of a graph G is a* **maximal** *connected subgraph of G.*

For a given graph $G(V, E)$,

a **clique** *is a subset $C \subseteq V$ such that any two vertices in C are adjacent;*

an **independent set** *is a subset $I \subseteq V$ such that no two vertices in I are adjacent;*

a **vertex cover** *is a subset $C \subseteq V$ such that for any edge $(u, v) \in E$, at least one of the endpoints is in I;*

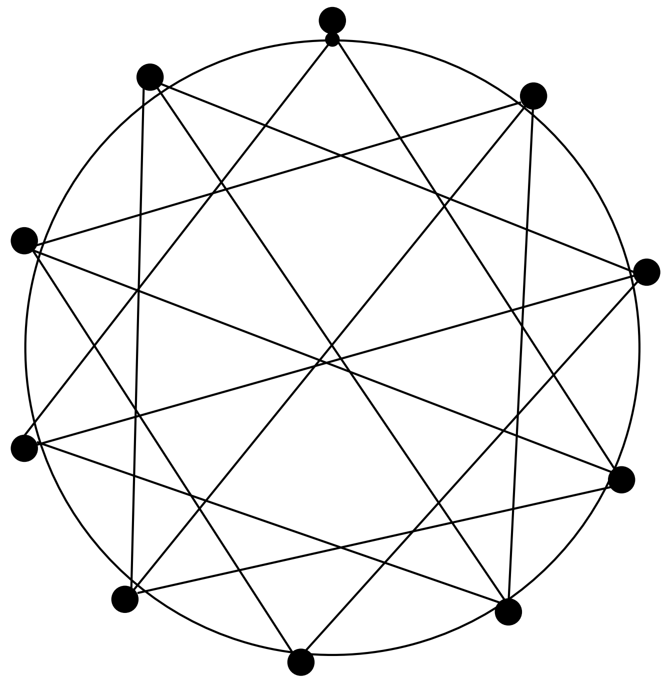
a **dominating set** *is a subset $D \subseteq V$ such that $\forall v \in V$, either $v \in D$, or v is adjacent to a vertex in D.*

Problem 1 *Prove that the following languages belong to $\mathcal{NP}$.*

Satisfiability: given a set of boolean variables $\{x_1, \ldots, x_n\}$, a set of *clauses* (a clause is a set of variables or their negations), is there an assignment to $\{x_i\}$ which makes all clauses true?

$$F = (\bar{x}_1 \vee \bar{x}_2) \wedge (\bar{x}_1 \vee x_2) \wedge (x_1 \vee \bar{x}_2) \wedge (x_1 \vee x_2)$$

Clique Problem: Given a graph $G(V, E)$ and an integer $k > 0$, does G have a clique of size $\geq k$?



IndSet Problem: Given a graph $G(V, E)$ and an integer $k > 0$, does G have an independent set of size $\geq k$.

Vertex Cover: Given a graph $G(V, E)$ and an integer $k > 0$, does G have a vertex cover of size $\leq k$.

Dominating Set: Given a connected graph $G(V, E)$ and an integer $k > 0$, does G have a dominating set of size $\leq k$.

Vertex Coloring: Given a graph $G(V, E)$ and an integer $k > 0$, is G k-colorable?

LCS: given k sequences $\{S_i\}_{i=1}^{k}$ and an integer t, is there a sequence C of length t which is a subsequence for every S_i.

5 NP-completeness

Definition 5 *Language A is* **polynomially mapping reducible** *or* **polynomially time reducible,** *to language B, if a polynomial time computable function $f : \Sigma^* \to \Sigma^*$ such that*

$$w \in A \quad \textit{iff} \quad f(w) \in B.$$

If A is polynomially time reducible to B, then we write

$$A \leq_P B.$$

Definition 6 *A language U is* **NP-complete**, *if*

1. *$U \in NP$; and*
2. *$\forall A \in NP,\ A \leq_P U$*

Theorem 2 *If $C \in NP$, U is NP-complete and $U \leq_P C$, then C is NP-complete.*

Theorem 3 *(Cook (1971) and Levin (1973))*
There are NP-complete problems.

SAT is NP-complete (*Cook (1971)*)