**Turing Machines**

Turing machine is the simplest model of computation.
- Input written on (infinite) one sided tape.
- Special blank characters.
- Finite state control (similar to DFA).
- Every step: Read character under head, write character out, move the head right or left (or stay).
- Every TM $M$ can be encoded as a string $⟨M⟩$.

Transition Function: $δ : Q \times Γ \to Q \times Γ \times \{←, →, □\}$

$δ(q,c) = (p,d,\rightarrow)$
- $q$: current state.
- $c$: character under tape head.
- $p$: new state.
- $d$: character to write under tape head.
- $\leftarrow$: Move tape head left.

---

**Complexity Classes**

*Algorithmic Complexity Classes (assuming $P \neq NP$)*

- Undecidable
- NP-Hard
- NPC
- NP
- co-NP
- PSPACE
- EXP

*Computational Complexity Classes*

- Turing-unrecognizable
  - Turing-unacceptable/recognizable, partially-decidable
- Semi-Decidable
  - ( recursively-enumerable, recognizable, Turing-acceptable/recognizable, partially-decidable
- Decidable
  - (Recursive
- Context-Sensitive
- Context-Free
- Regular

*Sample undecidable problems*

| ACCEPTONINPUT: $A_{TM} = \{⟨M, w⟩ : M$ is a TM and $M$ accepts on $w\}$ |
| HALTONINPUT: $Hal_{TM} = \{⟨M, w⟩ : M$ is a TM and halts on input $w\}$ |
| HALTONBLANK: $Hal_{B_{TM}} = \{⟨M⟩ : M$ is a TM & $M$ halts on blank input\} |
| EMPTYNESS: $E_{TM} = \{⟨M⟩ : M$ is a TM and $L(M) = \emptyset\}$ |
| EQUALITY: $E_{Q_{TM}} = \{⟨M_A, M_B⟩ : M_A and M_B are TM's and $L(M_A) = L(M_B)$\} |

---

**Reductions**

A general methodology to prove impossibility results.
- Start with some known hard problem $X$
- Reduce $X$ to your favorite problem $Y$

If $Y$ can be solved then so can $X \Rightarrow Y$. But we know $X$ is hard so $Y$ has to be hard too. On the other hand if we know $Y$ is easy, then $X$ has to be easy too.

The Karp reduction, $X \leq_P Y$ suggests that there is a polynomial time reduction from $X$ to $Y$.

Assuming
- $R(n)$: running time of $R$
- $Q(n)$: running time of $A_Y$

Running time of $A_X$ is $O(Q(R(n)))$

*Sample NP-complete problems*

- CIRCUITSAT: Given a boolean circuit, are there any input values that make the circuit output TRUE?
- 3SAT: Given a boolean formula in conjunctive normal form, with exactly three distinct literals per clause, does the formula have a satisfying assignment?
- INDEPENDENTSET: Given an undirected graph $G$ and integer $k$, is there a set of vertices $S \subseteq V(G)$ with size $|S| \leq k$ such that there are no edges between them?
- CLIQUE: Given an undirected graph $G$ and integer $k$, is there a complete subgraph of $G$ with size $\geq k$?
- kPARTITION: Given a set $X$ of $k$ positive integers, can $X$ be partitioned into $k$ subsets $S_1, S_2, \ldots, S_k$ such that each subset has the same sum?
- 3COLOR: Given an undirected graph, can its vertices be colored with three colors, so that every edge touches vertices with different colors?
- HAMILTONIANPATH: Given a graph $G$, is there a path that visits every vertex exactly once?
- HAMILTONIANCYCLE: Given a graph $G$, is there a cycle that visits every vertex exactly once?
- LONGESTPATH: Given a graph $G$ and an upper bound $B$, is there a path of length $\geq B$?

Remember a path is a sequence of distinct vertices $[v_1, v_2, \ldots, v_k]$ such that an edge exists between any two vertices in the sequence. A cycle is the same with the addition of a edge $(v_k, v_1) \in E$. A walk is a path except the vertices can be repeated.

A formula is in conjunctive normal form if variables are or’ed together inside a clause and then clauses are anded together inside a clause, then clauses are anded together inside clauses, and clauses are anded together inside clauses.

Disjunctive normal form is the opposite ($((x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5))$).