15-453

FORMAL LANGUAGES, AUTOMATA AND COMPUTABILITY

Read sections 7.1 – 7.3 of the book for next time

TIME COMPLEXITY OF ALGORITHMS

(Chapter 7 in the textbook)

COMPLEXITY THEORY

Studies what can and can't be computed under limited resources such as time, space, etc.

Today: Time complexity

MEASURING TIME COMPLEXITY

We measure time complexity by counting the elementary steps required for a machine to halt

Consider the language $A = \{ 0^k1^k \mid k \geq 0 \}$

2k **1. Scan across the tape and reject if the string is not of the form 0^m1^n**

$2k^2$ **2. Repeat the following if both 0s and 1s remain on the tape:**
Scan across the tape, crossing off a single 0 and a single 1

2k **3. If 0s remain after all 1s have been crossed off, or vice-versa, reject. Otherwise accept.**

- **The number of steps that an algorithm uses on a particular input may depend on several parameters.**
- **For instance, if the input is a graph, then the number of steps may depend on the number of nodes, the number of edges, *et cetera*.**
- **For simplicity, we compute the running time purely as a function of the length of the input string and don't consider any other parameters.**

Let M be a TM that halts on all inputs.

Assume we compute the running time purely as a function of the length of the input string.

Definition: The running time or time-complexity function of M is the function $f : N \rightarrow N$ such that f(n) is the maximum number of steps that M uses on any input of length n.

ASYMPTOTIC ANALYSIS

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

Big-O notation has been discussed in previous classes. We will briefly review it.

BIG-**O**

Let f and g be two functions $f, g : N \rightarrow R^+$.
We say that f(n) = O(g(n)) if positive integers c and n_0 exist so that for every integer $n \geq n_0$

$$f(n) \leq c\,g(n)$$

When f(n) = O(g(n)), we say that g(n) is an asymptotic upper bound for f(n)

$$5n^3 + 2n^2 + 22n + 6 = O(n^3)$$

If $c = 6$ and $n_0 = 10$, then $5n^3 + 2n^2 + 22n + 6 \leq cn^3$

$2n^{4.1} + 200283n^4 + 2 = O(n^{4.1})$

$3n \log_2 n + 5n \log_2 \log_2 n = O(n \log_2 n)$

$n \log_{10} n^{78} = O(n \log_{10} n)$

$\log_{10} n = \log_2 n / \log_2 10$

$O(n \log_2 n) = O(n \log_{10} n) = O(n \log n)$

Definition: TIME(t(n)) is the set of languages decidable in O(t(n)) time by a Turing Machine.

$\{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n^2)$

$A = \{ 0^k1^k \mid k \geq 0 \} \in \text{TIME}(n \log n)$

Cross off every other 0 and every other 1. If the # of 0s and 1s left on the tape is odd, reject.

000000000000011111111111

x0x0x0x0x0x0xx1x1x1x1x1x1x

xxx0xxx0xxx0xxxx1xxx1xxx1x

xxxxxxx0xxxxxxxxxxxx1xxxxx

xxxxxxxxxxxxxxxxxxxxxxxxx

We can prove that a (single-tape) TM can't decide *A* faster than O(n log n).

$A = \{ 0^k1^k \mid k \geq 0 \}$

Can $A = \{ 0^k1^k \mid k \geq 0 \}$ be decided in time O(n) with a two-tape TM?

- **Scan all 0s and copy them to the second tape.**
- **Scan all 1s, crossing off a 0 from the second tape for each 1.**

Different models of computation yield different running times for the same language!

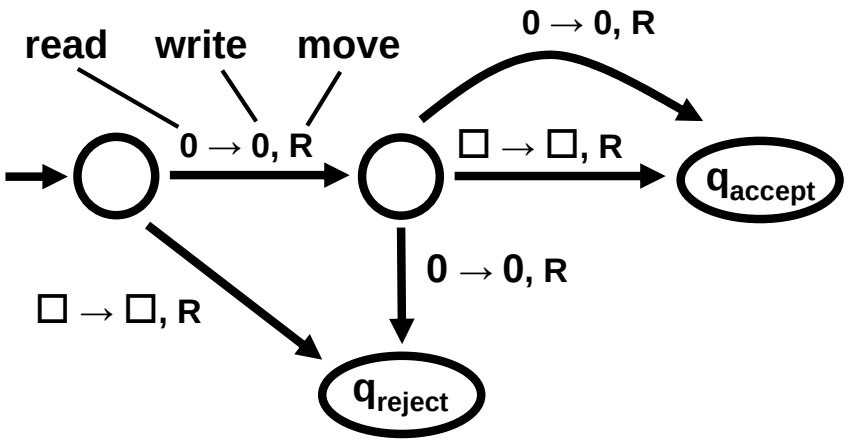
Theorem.
Let $t(n)$ be a function such that $t(n) \geq n$. Then every $t(n)$-time multi-tape TM has an equivalent $O(t(n)^2)$ single-tape TM.

Polynomial Time

$$P = \bigcup_{k \in N} TIME(n^k)$$

NON-DETERMINISTIC TURING MACHINES AND NP

NON-DETERMINISTIC TMs



NON-DETERMINISTIC TMs

…are just like standard TMs, except:

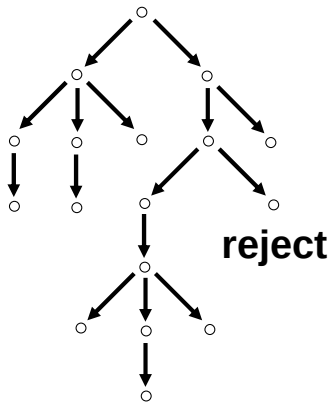
1. The machine may proceed according to several possibilities.

2. The machine accepts a string if there exists a path from start configuration to an accepting configuration.

Deterministic Computation

accept or reject

Non-Deterministic Computation



accept

Definition: A Non-Deterministic TM is a 7-tuple $T = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where:

Q is a finite set of states

Σ is the input alphabet, where $\square \notin \Sigma$

Γ is the tape alphabet, where $\square \in \Gamma$ and $\Sigma \subseteq \Gamma$

$$\delta : Q \times \Gamma \rightarrow \text{Pow}(Q \times \Gamma \times \{L,R\})$$

$q_0 \in Q$ is the start state

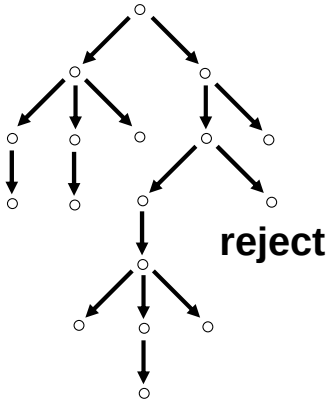
$q_{accept} \in Q$ is the accept state

$q_{reject} \in Q$ is the reject state, and $q_{reject} \neq q_{accept}$

Deterministic Computation

accept or reject

Non-Deterministic Computation



accept

Definition: NTIME(t(n)) is the set of languages decided by a O(t(n))-time non-deterministic Turing machine.

$$\text{TIME}(t(n)) \subseteq \text{NTIME}(t(n))$$

BOOLEAN FORMULAS

logical operations

parentheses

A satisfying assignment is a setting of the variables that makes the formula true.

$$\phi = (\neg x \wedge y) \vee z$$

x = 1, y = 1, z = 1 is a satisfying assignment for ϕ

variables

BOOLEAN FORMULAS

x = 0, y = 0, z = 1 is a satisfying assignment for:

$$\neg(x \vee y) \wedge (z \wedge \neg x)$$

0 0 1 0

0 1

1 1

A boolean formula is satisfiable if there exists a satisfying assignment for it.

YES $a \wedge b \wedge c \wedge \neg d$

NO $\neg(x \vee y) \wedge x$

Definition: SAT is the language consisting of all satisfiable boolean formulas.

SAT = { ϕ | ϕ is a satisfiable boolean formula }

Conjunctive Normal Form (CNF)

- **A literal is a variable or the negation of a var.**
  - **Example: The variable *x* is a literal, and its negation, $\neg x$, is a literal.**
- **A clause is a disjunction (an OR) of literals.**
  - **Example: $(x \vee y \vee \neg z)$ is a clause.**
- **A formula is in Conjunctive Normal Form (CNF) if it is a conjunction (an AND) of clauses.**
  - **Example: $(x \vee \neg z) \wedge (y \vee z)$ is in CNF.**
- **A CNF formula is a conjunction of disjunctions of literals.**

Definition: A CNF formula is a 3CNF-formula iff each clause has exactly 3 literals.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_2 \vee x_5) \wedge \ldots \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$$

clauses

- **A literal is a variable or the negation of a var.**
- **A clause is a disjunction (an OR) of literals.**
- **A formula is in Conjunctive Normal Form (CNF) if it is a conjunction (an AND) of clauses.**
- **A CNF formula is a conjunction of disjunctions of literals.**

Definition: A CNF formula is a 3CNF-formula iff each clause has exactly 3 literals.

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_2 \vee x_5) \wedge \ldots \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$

clauses

YES $(x_1 \vee \neg x_2 \vee x_1)$

NO $(x_3 \vee x_1) \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$

NO $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_4 \vee x_2 \vee x_1) \vee (x_3 \vee x_1 \vee \neg x_1)$

NO $(x_1 \vee \neg x_2 \vee x_3) \wedge (x_3 \wedge \neg x_2 \wedge \neg x_1)$

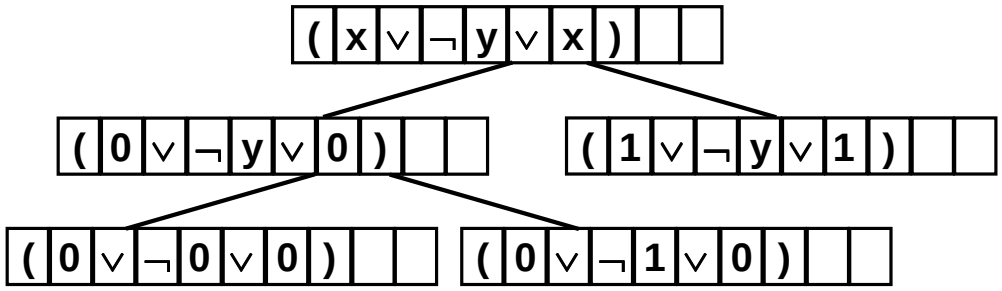
3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

3SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

Theorem: 3SAT $\in$ NTIME(n^2)

On input ϕ:

1. **Check if the formula is in 3cnf.**
2. **For each variable, non-deterministically substitute it with 0 or 1.**



3. **Test if the assignment satisfies ϕ.**

Non-deterministic Polynomial Time

$$NP = \bigcup_{k \in N} NTIME(n^k)$$

Theorem: L $\in$ NP if and only if there exists a poly-time Turing machine V with

L = { x | $\exists$y. |y| = poly(|x|) and V(x,y) accepts }.

Proof:

(1) If L = { x | $\exists$y. |y| = poly(|x|) and V(x,y) accepts } then L $\in$ NP.

Because we can guess y and then run V.

(2) If L $\in$ NP then

L = { x | $\exists$y. |y| = poly(|x|) and V(x,y) accepts }

Let N be a non-deterministic poly-time TM that decides L. Define V(x,y) to accept if y is an accepting computation history of N on x.

3SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ and ϕ is in 3cnf }

SAT = { ϕ | $\exists$y such that y is a satisfying assignment to ϕ }

A language is in NP if and only if there exist polynomial-length certificates for membership to the language.

SAT is in NP because a satisfying assignment is a polynomial-length certificate that a formula is satisfiable.

NP = The set of all the problems for which you can verify an alleged solution in polynomial time.

P = NP?

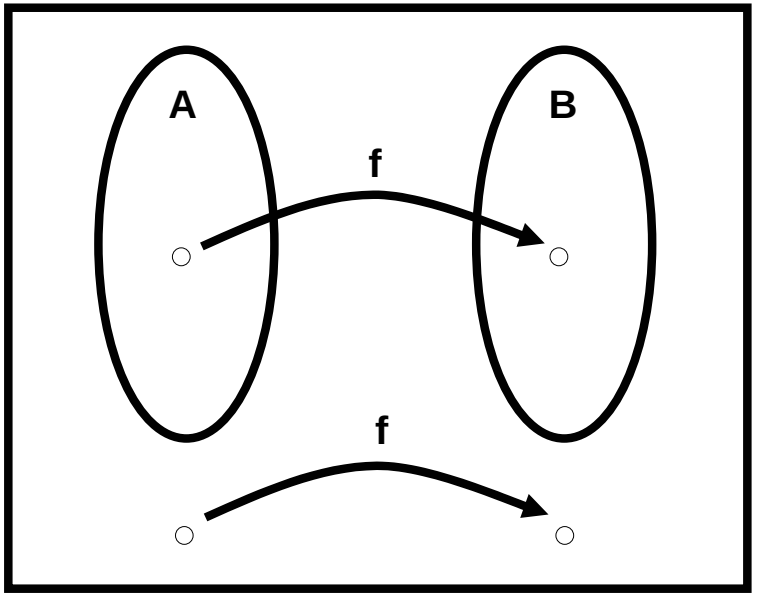
POLY-TIME REDUCIBILITY

$f : \Sigma^* \rightarrow \Sigma^*$ is a polynomial-time computable function if some poly-time Turing machine M, on every input w, halts with just f(w) on its tape.

Language A is polynomial time reducible to language B, written $A \leq_P B$, if there is a poly-time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that:

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a polynomial-time reduction of A to B.



Theorem: If $A \leq_P B$ and $B \in P$, then $A \in P$.

Proof: Let M_B be a poly-time (deterministic) TM that decides B and let f be a poly-time reduction from A to B.

We build a machine M_A that decides A as follows:

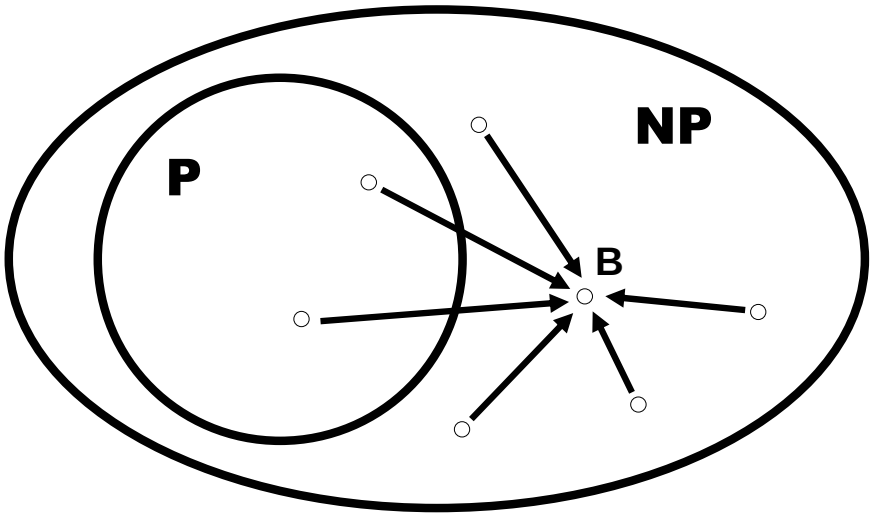
On input w:

1. **Compute f(w)**
2. **Run M_B on f(w)**

Definition: A language B is NP-complete iff:

1. **$B \in$ NP**
2. **Every language in NP is reducible to B in polynomial time.**

If B is NP-Complete and B $\in$ P then NP = P. Why?



SAT = { ϕ | ϕ is a satisfiable boolean formula }

3-SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

A 3cnf-formula is of the form:

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_4 \vee x_2 \vee x_5) \wedge (x_3 \vee \neg x_2 \vee \neg x_1)$

clauses

SAT = { ϕ | ϕ is a satisfiable boolean formula }

3-SAT = { ϕ | ϕ is a satisfiable 3cnf-formula }

SAT, 3-SAT $\in$ NP

Theorem (Cook-Levin): SAT is NP-complete.

Proof Outline:

(1) SAT $\in$ NP

(2) Every language A in NP is polynomial time reducible to SAT

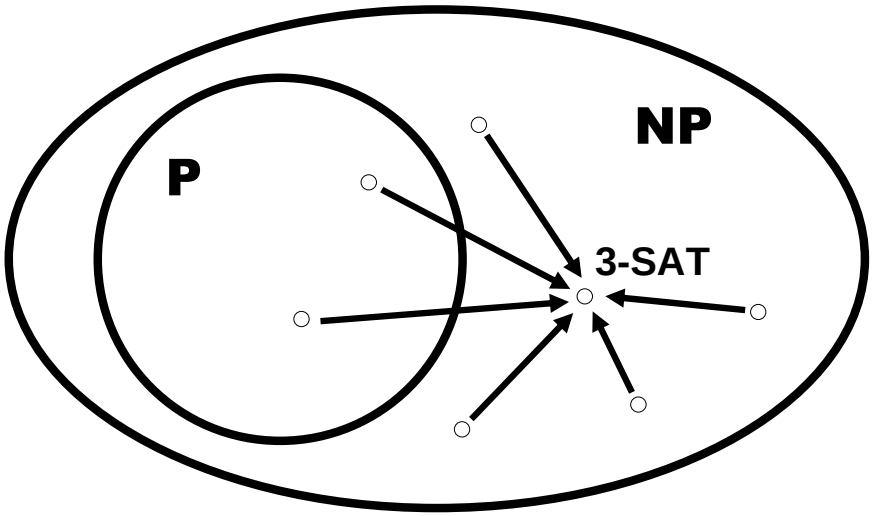
We build a poly-time reduction from A to SAT

The reduction turns a string w into a 3-cnf formula ϕ such that w $\in$ A iff $\phi \in$ 3-SAT.

ϕ will simulate the NP machine N for A on w.

Let N be a non-deterministic TM that decides A in time n^k How do we know N exists?

So proof will also show:
3-SAT is NP-Complete



f turns a string w into a 3-cnf formula ϕ such that

$w \in A \Leftrightarrow \phi \in$ 3-SAT.

ϕ will simulate an NP machine N on w, where A = L(N)