Intractable Problems

Time-Bounded Turing Machines

Classes **P** and **NP**

Polynomial-Time Reductions

Time-Bounded TM’s

- A Turing machine that, given an input of length n, always halts within T(n) moves is said to be *T(n)-time bounded*.
  - The TM can be multitape.
  - Sometimes, it can be nondeterministic.
- The deterministic, multitape case corresponds roughly to “an O(T(n)) running-time algorithm.”

The class **P**

- If a DTM M is T(n)-time bounded for some polynomial T(n), then we say M is *polynomial-time* ("*polytime*") bounded.
- And L(M) is said to be in the class **P**.
- Important point: when we talk of **P**, it doesn't matter whether we mean "by a computer" or "by a TM" (next slide).

Polynomial Equivalence of RAM algorithms and Turing Machine algorithms

- A multitape TM can simulate a RAM algorithm that runs for time O(T(n)) in at most $O(T^2(n))$ of its own steps.
- If T(n) is a polynomial, so is $T^2(n)$.

Examples of Problems in **P**

- Is w in L(G), for a given CFG G?
  - Input = w.
  - Use CYK algorithm, which is $O(n^3)$.
- Is there a path from node x to node y in graph G?
  - Input = x, y, and G.
  - Use BFS algorithm, which is O(n) on a graph of n nodes and arcs.

Running Times Between Polynomials

- You might worry that something like O(n log n) is not a polynomial.
- However, to be in **P**, a problem only needs an algorithm that runs in time less than some polynomial.
  - O(n log n) is less than $O(n^2)$.
- So, for most algorithms counting input size in bits or words is polynomial either way.

A Tricky Example: Partition

- The *Partition Problem*: given positive integers i_1, i_2 ,…, i_n, can we divide them into two sets with equal sums?
- We can solve this problem in by a dynamic-programming algorithm, but the running time is only polynomial if all the integers are relatively small.
  - Pseudo-polynomial time.

Subtlety: Measuring Input Size

- "Input size" has a specific meaning: the length of the representation of the problem instance as it is input to a TM.
- For the Partition Problem, you cannot always write the input in a number of characters that is polynomial in either the number-of or sum-of the integers.

Partition Problem – Bad Case

- Suppose we have n integers, each of which is around 2^n.
- We can write integers in binary, so the input takes $O(n^2)$ space to write down.
- But the dynamic programming solution would require time more than $n2^n$.

The Class **NP**

- The running time of a nondeterministic TM is the maximum number of steps taken along any branch.
- If that time bound is polynomial, the NTM is said to be *polynomial-time bounded*.
- And its language/problem is said to be in the class **NP**.

Example: **NP**

- The Partition Problem is definitely in **NP**, even using the conventional binary representation of integers.
- Use nondeterminism to guess one of the subsets.
- Sum the two subsets and compare to see if they are equal.

P Versus **NP**

- One of the most important open problems is the question **P** = **NP**?
- There are thousands of problems that are in **NP** but appear not to be in **P**.
- But no proof that they aren't really in **P**.
- Worth $1 million if you can solve it:
  - http://www.claymath.org/millennium-problems/p-vs-np-problem

Complete Problems

- One way to address the **P** = **NP** question is to identify *complete problems* for NP.
- A (decision) problem, L, is *NP-complete* if:

1. L is in **NP**.
2. Every problem in **NP** can be reduced to L in polynomial time (this will be formally defined shortly).

Complete Problems – Intuition

- A complete problem for a class embodies every problem in the class, even if it does not appear so.
- Compare: PCP embodies every TM computation, even though it does not appear to do so.
- Strange but true: Partition embodies every polytime NTM computation.

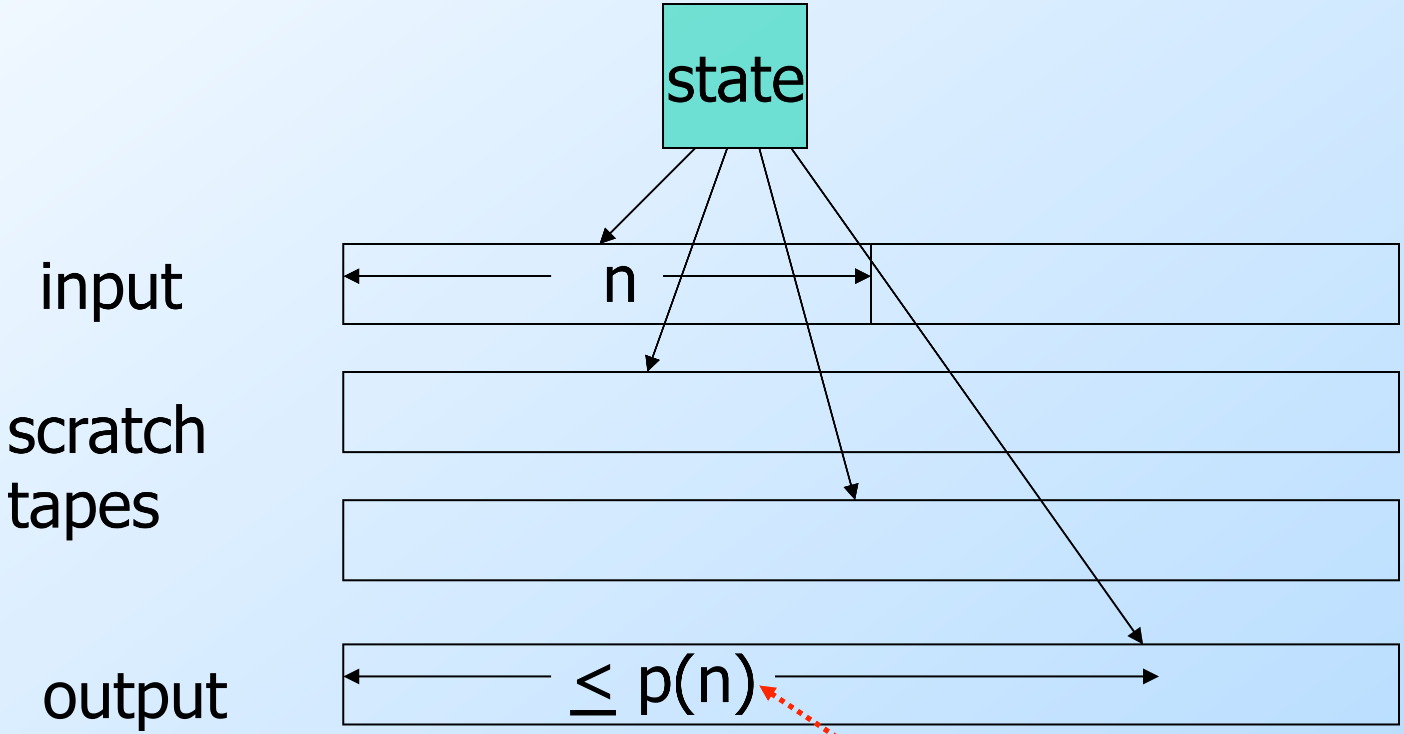
Polytime Reductions

- Goal: show a given problem, L, to be NP-complete by:
- First showing L is in **NP** (usually easy).
- Reducing every language/problem in **NP** to L in such a way that if we had a deterministic polytime algorithm for L, then we could construct a deterministic polytime algorithm for any problem in **NP**.

Polytime Reductions – (2)

- We need the notion of a *polytime transducer* – a TM that:
  1. Takes an input of length n.
  2. Operates deterministically for some polynomial time p(n).
  3. Produces an output on a separate *output tape*.
- Note: output length is at most p(n).

Polytime Transducer

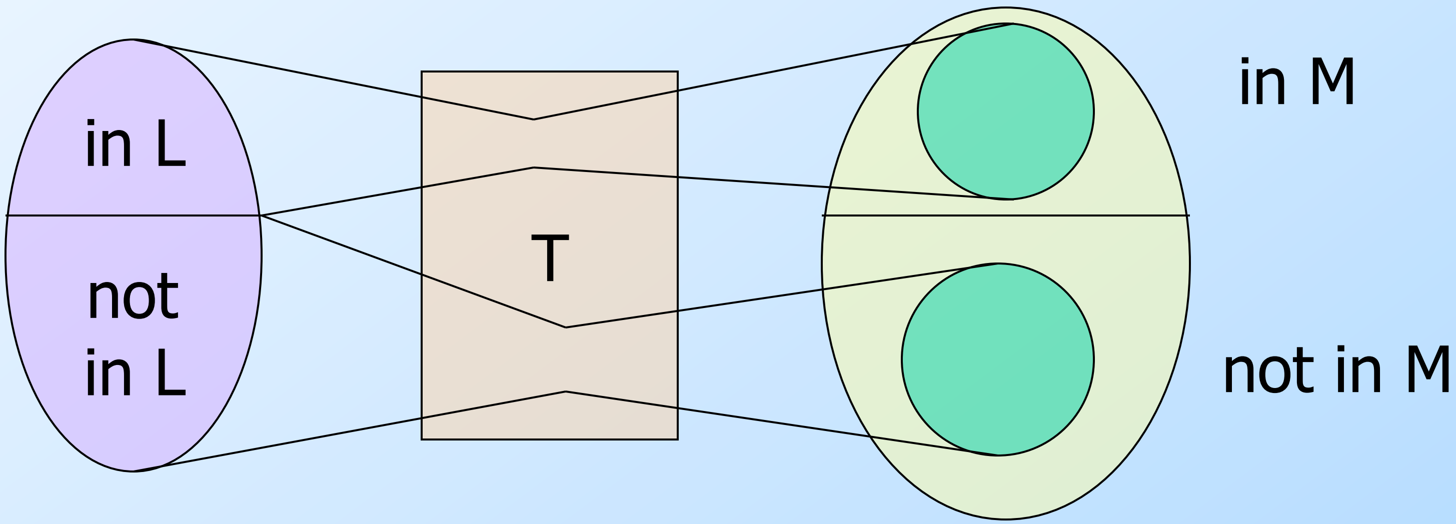


Remember: important requirement is that *time* ≤ p(n).

Polytime Reductions – (3)

- Let L and M be languages.
- Say L is *polytime reducible* to M if there is a polytime transducer T such that:
  - for every input w to T, the transducer's output, x = T(w), is in M if and only if w is in L.

Picture of Polytime Reduction



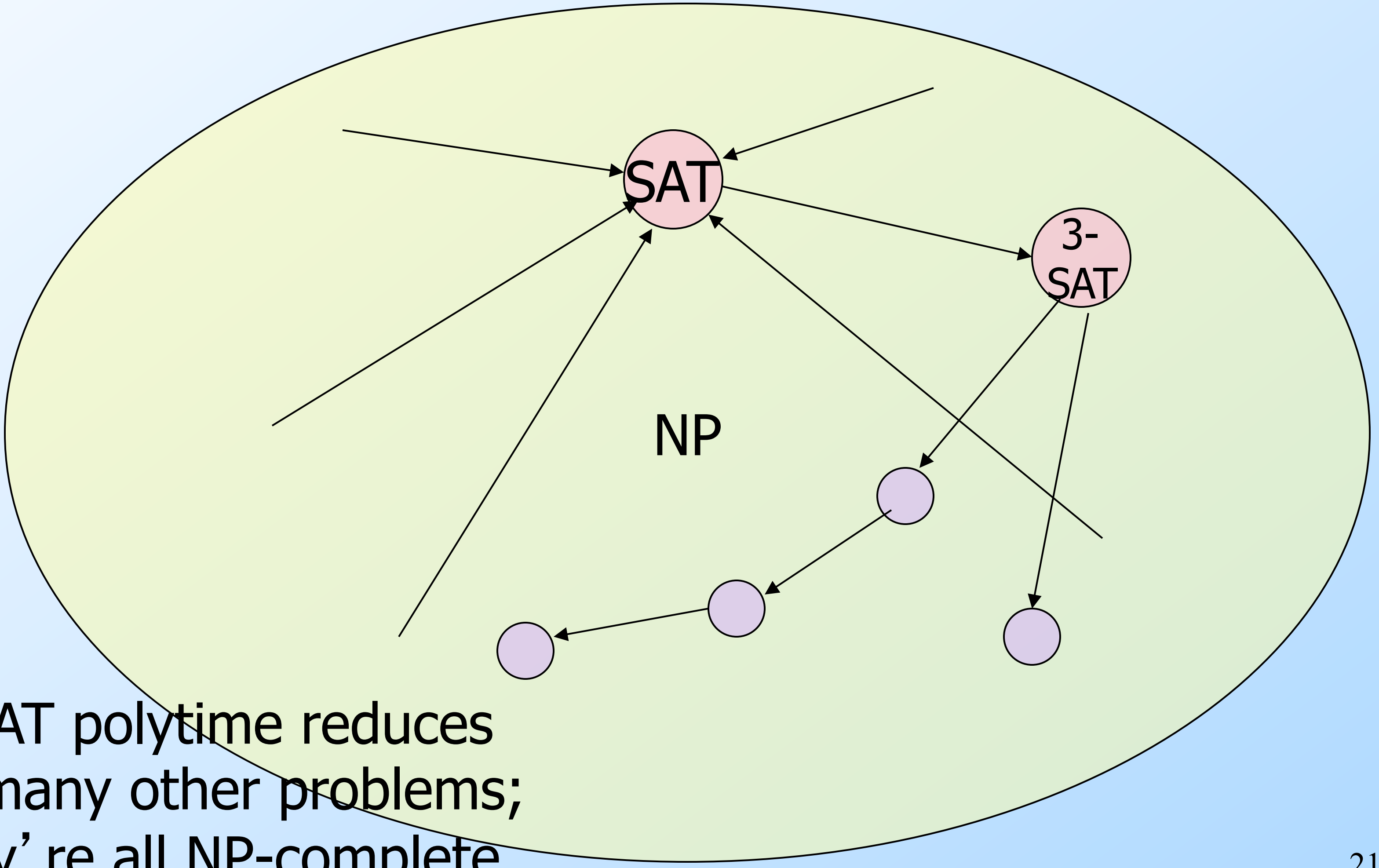
NP-Complete Problems

- A problem/language M is said to be *NP-complete* if M is in **NP** and, for every language L in **NP**, there is a polytime reduction from L to M.
- Fundamental property: if M has a polytime algorithm, then L also has a polytime algorithm.
  - I.e., if M is in **P**, then every L in **NP** is also in **P**, or "**P** = **NP**."

The Plan

All of **NP** polytime reduces to SAT, which is therefore NP-complete

SAT polytime reduces to 3-SAT



3-SAT polytime reduces to many other problems; they’re all NP-complete