COMPSCI 501: Formal Language Theory

Lecture 22: The Class NP

Marius Minea
marius@cs.umass.edu

University of Massachusetts Amherst

20 March 2019

Recap: Time Complexity

Time complexity class TIME$(t(n))$ = all languages that are decidable by an $O(t(n))$ (deterministic, single-tape) Turing machine.

A $t(n)$ **multitape** TM has an equivalent $O(t^2(n))$ single-tape TM.

multi-tape *polynomial* ⇒ single-tape *polynomial*

Every $t(n)$ **nondeterministic** TM has an equivalent $2^{O(t(n))}$ deterministic single-tape TM.

nondeterministic *polynomial* ⇒ single-tape *exponential*

general-case construction, could be better in some cases

Checking vs. Proving vs. Disproving

Hamiltonian Path: directed path containing each node once

Decidable ⇒ complement is decidable

- Finding: hard (NP-complete, will revisit)
- Checking: easy (traverse path, mark all graph nodes)
  Complexity: polynomial $O(n^2)$
- Disproving: hard (try all paths)

- Checking witness easier than finding
- Complement may not have witness

Polynomial Verifiers

Def. A **verifier** for a language A is an algorithm V, where

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

A **polynomial-time verifier** runs polynomial in the length of v.

A language is **polynomially verifiable** if it has a polynomial time verifier.

Why no specified constraint on the certificate c?

If verifier is polynomial, certificate must be polynomial too, otherwise no time to read the entire certificate!

The Class NP

Def. **NP** is the class of languages that have polynomial-time verifiers.

For *HAMPATH*, the certificate can be simply the path.

$$\textit{COMPOSITES} = \{x \mid x = pq, \text{ for integers } p, q, > 1\}$$

Certificate c: a divisor of x. ⇒ *COMPOSITES* is in NP.

Certificate for *PRIMES*: ??? (negation is asymmetric)

We now know *PRIMES* is in P (Agrawal-Kayal-Saxena 2002) ⇒ *COMPOSITES* is also in P

NP = Nondeterministic Polynomial Time

Theorem: A language is in NP iff it is decided by some nondeterministic polynomial time Turing machine

polynomial verifier ⇔ polynomial-time NTM

"⇒" Assume verifier V running in time n^k.

NTM N is:

1. Nondeterministically choose string c of length $< n^k$
   (n^k steps, Σ branches at each)
2. Run V on input $\langle w, c\rangle$ (poly-time)
3. Accept/reject based on V

"⇐" Construct verifier that treats each symbol of c as a description of the nondeterministic choice to make by N.

1. Simulate N on w according to choices c
2. If this branch of N accepts, accept, else reject

The Complexity Class NTIME

Def. **NTIME($t(n)$)** $= \{L \mid L$ is a language decided by an $O(t(n))$ time nondeterministic Turing machine $\}$

$$\text{NP} = \bigcup_k \text{NTIME}(n^k)$$

Showing that a problem is in NP

- Construct a poly-time verifier (certificate c is usually the solution) or
- Have a NTM nondeterministically generate c and then check it

CLIQUE

A *clique* of an undirected graph is a subgraph with all nodes connected.

Does graph G have a complete subgraph with k nodes ?

Certificate: set of nodes that should form clique.

SUBSET-SUM

Given a collection (multiset) of integers, and a target integer t, is there a subcollection of numbers adding up to t ?

Certificate: the subset which should add up to t

The class co-NP

For *CLIQUE* and *SUBSET-SUM*, no obvious way to quickly verify a NO answer
can't say that the **complements** of these sets are in NP
(no obvious certificate for *complement* of these problems)

co-NP: languages whose complements are in NP.

co-NP $\neq$ NP ?? – don't know

Examples:

Is a formula valid? counterexample: falsifying assignment
this is actually $\overline{SAT}$

PRIMES is in co-NP: counterexample: proper divisor
we know it's also in P

P versus NP

Recall: decide vs. verify

P = can *decide* membership in polynomial time

NP = can *verify* membership in polynomial time (given certificate)

Trivially, P $\subseteq$ NP. We may have P = NP or P $\subset$ NP

We've seen poly-time NTM $\Rightarrow$ exponential-time DTM. Thus

$$\text{NP} \subseteq \text{EXPTIME} = \bigcup_k \text{TIME}(2^{n^k})$$

We don't know if there is a stronger deterministic-time bound.

Polynomial-time Reductions

Def. A function $f : \Sigma^* \to \Sigma$ is a **polynomial time computable function** if some polynomial time Turing machine exists that when started with any w, halts with just $f(w)$ on the tape.

Def. Language A is **polynomial-time (mapping) reducible** to language B ($A \leq_P B$) if a polynomial-time computable function $f : \Sigma^* \to \Sigma^*$ exists, where for all w,

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

- Reduction (function) goes one-way (construct B-problem from A)
- Equivalence proof goes **both** ways
  YES maps to YES not enough
  also need NO mapped to NO