Computer Science 511

Nondeterministic Time: More about $\mathcal{NP}$— and co-$\mathcal{NP}$

Instructor: Wayne Eberly

Department of Computer Science
University of Calgary

Lecture #10

Goals for Today

Goals for Today

- Review fundamentals of the theory of $\mathcal{NP}$-completeness.
- Introduce the complexity class co-$\mathcal{NP}$ — along with a conjecture about that, and a consequence of that conjecture.

$\mathcal{NP}$-Completeness

- ***Definition:*** A language $L \subseteq \Sigma^\star$ is ***$\mathcal{NP}$-Hard*** (or, respectively ***$\mathcal{NP}$-Complete***) it is hard (respectively, complete) for $\mathcal{NP}$ with respect to polynomial-time many-one reductions.
- One can also consider languages that are hard (respectively, complete) with respect to polynomial-time oracle reductions. A reason, why the first definitions of hardness (respectively, completeness) are generally preferred, will be given later on in these notes.

An $\mathcal{NP}$-Complete Language

Consider the language $A_{\text{NTM+I+Time}}$ — consisting of encodings of nondeterministic multi-tape Turing machines M, input strings ω for M, and positive integers T (encoded in unary) such that M accepts ω using at most T steps — introduced in Lecture #9.

Claim #1: $A_{\text{NTM+I+Time}} \in \mathcal{NP}$.

An $\mathcal{NP}$-Complete Language

How This Can Be Proved: Consider a verification algorithm such that

- The certificate alphabet is the same as the input alphabet, Σ_{UTM}.
- A ***certificate*** for a string $\mu \in A_{\text{NTM+I+Time}}$, encoding a nondeterministic Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}}),$$

  input string $\omega \in \Sigma^\star$, and positive integer T, is an encoding ζ of a sequence of t moves, where $t \leq T$, that are consistent with M's transition function δ, going from the initial configuration for ω to an accepting configuration. Such a certificate exists, and $|\zeta| \leq |\mu|^2$ for every such certificate ζ.

An $\mathcal{NP}$-Complete Language

- Given an input that includes a string μ (such that we wish to check whether $\mu \in A_{\text{NTM+I+Time}}$) and a string ζ (that might be a certificate), a verification algorithm could proceed as follows.

    1. Check whether $\mu \in L_{\text{NTM+I+Time}}$, for the language $L_{\text{NTM+I+Time}}$ introduced in Lecture #9) — ***rejecting*** if this is not the case.

       Suppose, now, that μ encodes a nondeterministic Turing machine M, input string ω for M, and a positive integer T.
    2. Check whether $|\zeta| \leq |\mu|^2$ — ***rejecting*** if this is not the case.
    3. Check whether ζ encodes a sequence of at most T moves that are consistent with M's transition function, δ — ***rejecting*** if this is not the case.
    4. Apply the steps used by the "nondeterministic universal Turing machine", from Lecture #9, to continue (*after* an encoding of a sequence of moves has been guessed) — noting that the remaining steps are deterministic, and can be used to complete a validation algorithm. $\square$

An $\mathcal{NP}$-Complete Language

Claim #2: Let $L \subseteq \Sigma^\star$, for an alphabet Σ, such that $L \in \mathcal{NP}$. Then $L \preceq_{\text{P, M}} A_{\text{TM+I+Time}}$.

Sketch of Proof:

- Since $L \in \mathcal{NP}$ there exists a nondeterministic Turing machine

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$$

  and non-negative integers c_0, c_1 and d such that M decides L, and M's computation tree for ω has depth at most $c_1 \cdot |\omega|^d + c_0$ for every string $\omega \in \Sigma^\star$.

An $\mathcal{NP}$-Complete Language

- Since M is a ***fixed*** nondeterministic Turing machine, its shortest encoded is a ***fixed*** string in $\Sigma_{\text{UTM}}^{\star}$.
- The function

$$f : \Sigma^{\star} \to \Sigma_{\text{UTM}}^{\star}$$

  mapping each string $\omega \in \Sigma^{\star}$ to the encoding of M, ω (as an input for M) and the time bound $c_1 \cdot |\omega|^d + c_0$, can be shown to be a polynomial-time many-one reduction from L to $A_{\text{NTM+I+Time}}$ — establishing that $L \preceq_{\text{M}} A_{\text{NTM+I+Time}}$, as claimed. □

An $\mathcal{NP}$-Complete Language

- It follows by Claim #1, Claim #2, and the definition of "$\mathcal{NP}$-complete" that the language $A_{\text{NTM + Time + I}}$ is $\mathcal{NP}$-complete.
- Unfortunately, this is not very helpful because it is not clear how one can use this information to show that any *other* languages are $\mathcal{NP}$-complete, too.
- Lectures #10 and #11 review results establishing the existence of a *large* collection of $\mathcal{NP}$-complete languages.
- This makes the ***process*** to establish the $\mathcal{NP}$-completeness of a given language — described next — easier to use.

Proving $\mathcal{NP}$-Completeness of a Given Language

Suppose we want to prove that a given language $L \subseteq \Sigma^\star$ is $\mathcal{NP}$-complete. Then the following ***process*** can be used to do this — noting that steps #1 and #2, below, can be carried out in either order.

1. Prove that $L \in \mathcal{NP}$. This is, generally, accomplished by describing a ***certificate*** for a string in L, along with a ***polynomial-time verification algorithm*** for this language and set of certificates.
2. Prove that L is $\mathcal{NP}$-hard — by choosing some other language $\widehat{L} \subseteq \widehat{\Sigma}^\star$, that is already known to be $\mathcal{NP}$-complete, and then describing (and proving correctness of) a polynomial-time many-one reduction from $\widehat{L}$ to L.

It will then follow, by Corollary #16 in Lecture #7, that L is $\mathcal{NP}$-complete.

Proving $\mathcal{NP}$-Completeness of a Given Language

Note:

- Both of these steps require that an algorithm be presented and proved to be both correct and asymptotically efficient. ***Refinement*** — a process where you describe, and establish the correctness and efficiency of a "high-level" algorithm, and then gradually add detail in a correctness- and efficiency-preserving way — will often be useful.
- The choice of the $\mathcal{NP}$-complete language $\widehat{L}$, used in Step #2, can be very important! Describing a polynomial-time many-one reduction, from this language to L, can be significantly simpler for some choices of this language than it is for others.

The Complexity Class co-$\mathcal{NP}$

Recall that if $L \subseteq \Sigma^\star$ then the ***complement*** of L is the language

$$L^C = \{\omega \in \Sigma^\star \mid \omega \notin L\}.$$

Some references denote this by $\overline{L}$ instead of L^C.

It is easy to prove that $L \in \mathcal{P}$ if and only $L^C \in \mathcal{P}$, for every language $L \subseteq \Sigma^\star$.

It is ***not*** known whether this is also true for $\mathcal{NP}$, but most computer scientists (with an opinion) believe that it is not.

The Complexity Class co-$\mathcal{NP}$

Definition: co-$\mathcal{NP} = \{L^C \mid L \in \mathcal{NP}\}$.

Conjecture: $\mathcal{NP} \neq$ co-$\mathcal{NP}$.

The following is easy to prove.

Claim #3:

(a) $\mathcal{P} \subseteq \mathcal{NP} \cap$ co-$\mathcal{NP}$.

(b) If $\mathcal{P} = \mathcal{NP}$ then $\mathcal{NP} =$ co-$\mathcal{NP}$.

(c) If either $\mathcal{NP} \subseteq$ co-$\mathcal{NP}$ or co-$\mathcal{NP} \subseteq \mathcal{NP}$ then $\mathcal{NP} =$ co-$\mathcal{NP}$.

It is not known whether $\mathcal{P} = \mathcal{NP} \cap$ co-$\mathcal{NP}$.

A Useful Closure Property

The following is also easy to prove.

Claim #4:

(a) The complexity class $\mathcal{NP}$ is closed under polynomial-time many-one reductions.

(b) For all languages $L_1 \subseteq \Sigma_1^\star$ and $L_2 \subseteq \Sigma_2^\star$, if $L_1 \preceq_{\text{P, M}} L_2$ then $L_1^C \preceq_{\text{P, M}} L_2^C$.

(c) The complexity class co-$\mathcal{NP}$ is closed under polynomial-time many-one reductions.

The Complexity Class co-$\mathcal{NP}$

Definition: A language $L \subseteq \Sigma^\star$ is **co-$\mathcal{NP}$*-hard*** (respectively **co-$\mathcal{NP}$*-complete***) if L is hard (respectively, complete) for co-$\mathcal{NP}$ with respect to polynomial-time many-one reductions.

Claim #5: For every language $L \subseteq \Sigma^\star$,

(a) L is $\mathcal{NP}$-hard if and only if L^C is co-$\mathcal{NP}$-hard.

(b) L is $\mathcal{NP}$-complete if and only if L^C is co-$\mathcal{NP}$-complete.

It follows from this that, as languages are proved to be $\mathcal{NP}$-complete, related languages are established as co-$\mathcal{NP}$-complete with no extra work.

The Complexity Class co-$\mathcal{NP}$

As previously noted it is conjectured — but known — that polynomial-time oracle reductions and polynomial-time many-one reductions are not the same. As the following result shows, this conjecture is related to conjecture about the relationship between $\mathcal{NP}$ and co-$\mathcal{NP}$ that is given above.

Claim #6: If $\mathcal{NP} \neq$ co-$\mathcal{NP}$ then there exist languages $L_1, L_2 \subseteq \Sigma^\star$ (for an alphabet Σ) such that $L_1 \preceq_{\text{P, O}} L_2$, but $L_1 \not\preceq_{\text{P, M}} L_2$.