INFR11102: Computational Complexity 09/10/2018

Lecture 7: More on $\mathtt{NL}$; $\mathtt{NP}$-completeness

Lecturer: Heng Guo

1 More on $\mathtt{NL}$

Last time we have shown that Conn is $\mathtt{NL}$-complete. It implies that if Conn $\in \mathtt{L}$, then $\mathtt{NL} = \mathtt{L}$.

A natural variant of Conn is the undirected graph version.

Name: UConn

Input: A undirected graph G with two vertices s and t.

Output: Are s and t connected?

This is the complete problem for a class called $\mathtt{SL}$ (symmetric log-space), for which the formal definition is quite complicated. In fact, the motivation of defining $\mathtt{SL}$ [LP82] is to find a complexity class to place UConn. Two decades later, an amazing result by Omer Reingold [Rei08] shows that UConn $\in \mathtt{L}$ and thus $\mathtt{SL} = \mathtt{L}$. (Very roughly speaking, it is achieved by derandomizing random walks.)

However, we are still unable to resolve whether $\mathtt{NL} \overset{?}{=} \mathtt{L}$.

1.1 $\mathtt{NL} = \mathtt{coNL}$ (non-examinable)

Another variant of Conn, namely its complement, is the following.

Name: $\overline{\text{Conn}}$

Input: A directed graph G with two vertices s and t.

Output: Is there *no* path in G from s to t?

Neil Immerman [Imm88] and Róbert Szelepcsényi [Sze88] independently proved the following theorem.

Theorem 1. $\overline{\text{Conn}} \in \mathtt{NL}$.

We will not prove the theorem here, but we should step back and take a minute to think about the meaning of this theorem.

A naive $\mathtt{NL}$ algorithm for $\overline{\text{Conn}}$ is to take the same NTM for Conn but flip its answers. Thus, we will accept, if there exists a wrong guess connecting s to t. However, for any (s, t), there exists a wrong guess, and we will end up accepting everything!

Essentially, if (G, s, t) is in $\overline{\textsc{Conn}}$, then for any sequence of vertices of G, it cannot connect s to t. There is a universal quantifier "for any" in the statement. However, recall the verification definition of $\mathtt{NP}$, what an NTM is capable of doing is to state an existential quantifier "there exists ...". Thus, Theorem 1 is non-trivial in that it implicitly transforms the universal quantifier into an existential one.

Without the space requirement, this can actually be done easily. For example, if there exists a *cut*[1] (X, Y) of G, such that $s \in X$ and $t \in Y$, then we know that s and t are disconnected. However, a cut cannot be stored in logarithmic space. The proof of Theorem 1 uses a method called *inductive counting*.

The idea is that, if we know how many vertices are reachable from s, then we can simply guess connectivity for every vertex, verify the guess, and then verify the total number is correct. The ability to verify the guess makes sure that we never overestimate the number of reachable vertices. Hence, if the total number in the end is correct, then we know all our guesses are correct, and thus know whether t is among reachable vertices. To find out this number, let C_k be the number of vertices that can be reached within k steps. To compute C_n, we do it inductively from C_1 to C_n. If we know C_k, then we can verify whether any vertex $v \in V$ can be reached from s within k steps by guessing and checking. Thus we can use guessing and checking again to check whether any vertex $v \in V$ can be reached from s within $k+1$ steps. Thus we can get the count C_{k+1} and we inductively continue. This is a very non-intuitive algorithm and it exploits the power of non-determinism heavily. Full proof details can be found in [AB09, Theorem 4.20], [Pap94, Theorem 7.6], or the Supplementary Note on the course website.

The class of negations of languages from a complexity class $\mathtt{C}$ is often denoted $\mathtt{co}\cdot\mathtt{C}$. Formally, denote by $\overline{L}$ the complement language of L, and

$$\mathtt{coNL} := \{L \mid \overline{L} \in \mathtt{NL}\}.$$

It is easy to see that $\overline{\textsc{Conn}}$ is complete for $\mathtt{coNL}$. Thus, Theorem 1 implies that $\mathtt{coNL} = \mathtt{NL}$. However, it is still an important open question whether $\mathtt{NP} \overset{?}{=} \mathtt{coNP}$.

We note that $\mathtt{co}\cdot\mathtt{C}$ is not the complement of $\mathtt{C}$ in the set-theoretic sense, namely $\mathtt{co}\cdot\mathtt{C} \neq \overline{\mathtt{C}}$. Note that $\mathtt{C} \cap \overline{\mathtt{C}} = \emptyset$, but $\mathtt{NL} = \mathtt{coNL}$ (by Theorem 1) and $\mathtt{P} \subseteq \mathtt{NP} \cap \mathtt{coNP}$. Also, trivially, $\mathtt{co}\cdot\mathtt{P} = \mathtt{P}$ by flipping the accepting / rejecting states.

2 $\mathtt{NP}$-completeness

One of the most important technique in complexity theory is reductions. We have seen log-space reductions $\leq_\ell$ last time.

For problems in $\mathtt{NP}$, we will consider the following notion, called *Karp reductions* or *polynomial-time many-one reductions*.

Definition 1 (Karp reductions). *A language A is Karp reducible to another language B, denoted $A \leq_p B$, if there is a function $f : \Sigma^* \to \Sigma^*$ such that,*

[1] A partition (X, Y) of the vertices so that there is no edge from X to Y.

- *$x \in A \Leftrightarrow f(x) \in B$;*
- *f can be computed in polynomial time.*

By Definition 1, if $A \leq_p B$ and $B \in \texttt{P}$, then $A \in \texttt{P}$. Also, if $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$.

We now proceed to define $\texttt{NP}$-completeness.

Definition 2 ($\texttt{NP}$-completeness)**.** *A language A is* $\texttt{NP}$-hard *(under Karp reductions) if for any $B \in \texttt{NP}$, $B \leq_p A$.*

A is $\texttt{NP}$-complete *if $A \in \texttt{NP}$ and A is $\texttt{NP}$-hard.*

$\texttt{NP}$-complete problems are the most difficult ones among all problems in $\texttt{NP}$. By Definition 2, if there exists a $\texttt{NP}$-complete language $A \in \texttt{P}$, then $\texttt{P} = \texttt{NP}$!

Definition 2 sounds like very demanding, but $\texttt{NP}$-complete problems do exist. The canonical one is the following:

Name: $L_{\texttt{NP}}$

Input: A non-deterministic TM N, an input x, and a unary string 1^t.

Output: Does N accept x within t steps?

Proposition 2. *$L_{\texttt{NP}}$ is $\texttt{NP}$-complete.*

Proof. It is easy to see that $L_{\texttt{NP}} \in \texttt{NP}$. Given N, x and 1^t, we non-deterministically simulate N on x for t steps and output the same bit.

Next we show $L_{\texttt{NP}}$ is $\texttt{NP}$-hard. Let $A \in \texttt{NP}$ be a language computed by a NTM N_A in time $c_1 n^{c_2}$ for constants c_1 and $c_2 \geq 1$. Our reduction algorithm, given x, simply outputs $(N_A, x, 1^{c_1 n^{c_2}})$. Then let us check Definition 1:

- $x \in A$ if and only if $(N_A, x, 1^{c_1 n^{c_2}}) \in L_{\texttt{NP}}$ by the definition of $L_{\texttt{NP}}$;
- the reduction takes time $O(n^{c_2})$ (mostly to write down $1^{c_1 n^{c_2}}$). □

The 1^t part of the input is to make sure that $L_{\texttt{NP}}$ is in $\texttt{NP}$. If the input t was in binary, namely its size is $\log t$, then the simulation would take some polynomial time in t, which is exponential in the input size, $\log t$.

It is somewhat straightforward to design complete languages like this for other complexity classes, such as $\texttt{PSpace}$ or $\texttt{L}$, as well.

Remark (Bibliographic). Relevant chapters are [AB09, Chapter 2] and [Pap94, Chapter 8 and 9].

References

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

[LP82] Harry R. Lewis and Christos H. Papadimitriou. Symmetric space-bounded computation. *Theor. Comput. Sci.*, 19:161–187, 1982.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4):17:1–17:24, 2008.

[Sze88] Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, 1988.