**E0 224: Computational Complexity Theory** **Fall 2015**

Lecture 7: Aug 27, 2015

*Lecturer: Chandan Saha* *Scribe: Saravanan K*

## 7.1 Recap

In the last lecture we have studied various complexity classes such as **SPACE**, **NSPACE**, **PSPACE**, **NPSPACE**, **L**, **NL**. We also proved (Savitch's theorem) that $\mathbf{NSPACE}(S(n)) = \mathbf{SPACE}(S(n)^2)$, if $S(n) \geq \log n$.

The primary focus of this lecture is about **PSPACE** and **PSPACE** - COMPLETENESS.

**Claim 7.1 :** $\mathbf{SPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$

**Proof:** Let $L \in \mathbf{SPACE}(S(n))$ and $M$ be the corresponding Turing Machine that decides $L$. As we know, we denote the configuration graph for $M$ with input $\mathbf{x}$ as $G_{M,\mathbf{x}}$. Now,

$$|G_{M,\mathbf{x}}| = 2^{O(S(n))} \qquad , \textit{ where } n = |\mathbf{x}|$$

We check whether $C_{accept}$ is reachable from $C_{start}$ in $G_{M,\mathbf{x}}$ using a Deterministic Turing machine that runs in time $2^{O(S(n))}$. Hence $L \in \mathbf{DTIME}(2^{O(S(n))})$.

## 7.2 PSPACE - Completeness

The question, whether $\mathbf{P} = \mathbf{PSPACE}$ motivates us to study a new class $\mathbf{PSPACE} - \textit{Complete}$.

**Defn :** A language $L \in \{0,1\}^*$ is $\mathbf{PSPACE} - \textit{Complete}$ if

1. $L \in \mathbf{PSPACE}$ and
2. for all $L^{'} \in \mathbf{PSPACE}$, $L^{'} \leq_p L$.

Intuitively (Informally), $\mathbf{PSPACE} - \textit{complete}$ problems are the set of hardest problems in **PSPACE**.

**Claim 7.2 :** If $L$ is $\mathbf{PSPACE} - \textit{complete}$ and $L \in \mathbf{P}$, then $\mathbf{P} = \mathbf{PSPACE}$.
**Proof :** Since $L$ is $\mathbf{PSPACE} - \textit{complete}$, all problems $L^{'} \in \mathbf{PSPACE}$ reduces to $L$ in polynomial time. Given input $\mathbf{x}$ we map it to an instance $f(\mathbf{x})$, where $f$ is the polynomial function that reduces $L^{'}$ to $L$. We know $\mathbf{x} \in L^{'}$ iff $f(\mathbf{x}) \in L$ and $L \in \mathbf{P}$. This implies $f(\mathbf{x}) \in L$ can be determined in polynomial time and hence $L^{'} \in \mathbf{P}$.

**Example :** Let $L = \{(M, \mathbf{x}, 1^m) : M$ accepts $\mathbf{x}$ using at most $O(m)$ space$\}$. Is $L \in$ **PSPACE**-*complete* ?
**Answer :** YES.
**Proof :** a) $L \in \mathbf{PSPACE}$ : Given input $\mathbf{y}$ of the form $(M, \mathbf{x}, 1^m)$ there exists a universal Turing machine $M_U$ that simulates $M$ on input $\mathbf{x}$. As $M_U$ uses constant space overhead, $\mathbf{y} \in L$ iff $M_U$ uses $O(m)$ space. Therefore $L \in \mathbf{PSPACE}$.

b) $L \in$ **PSPACE**$-$*complete* : For a language $L^{'} \in$ **PSPACE**, there exists a Turing machine $M$ that decides $L^{'}$ using $p(n)$ space, where $p(n)$ is a polynomial function. Let $\mathbf{x}$ be the input for $M$.

We map $\mathbf{x} \mapsto (M, \mathbf{x}, 1^{p(|\mathbf{x}|)})$ in time $O(p(|x|))$. By definition of $L$,

$$(M, \mathbf{x}, 1^{p(|\mathbf{x}|)}) \in L \iff \mathbf{x} \in L^{'}$$

Hence the given language $L$ is **PSPACE** $-$ *Complete*.

## 7.3 Quantified Boolean Formulae (QBF)

**Definition (QBF)** QBF is a formula of the form $Q_1 x_1 \ Q_2 x_2 \ Q_3 x_3 \cdots \ Q_n x_n \ \phi(x_1, x_2, x_3, \cdots, x_n)$, where each quantifier $Q_i$ is either $\exists$ or $\forall$ and $\phi(x_1, x_2, \cdots, x_n)$ is a boolean formula.

**Example :** Consider the QBF $\exists x_1 \forall x_2 \ (\neg x_1 \vee x_1 x_2)$. The given QBF is true, because there exists an $x_1$ $(x_1 = 0)$ that makes the formula true for all $x_2$ $(x_2 \in 0, 1)$.

**Remark :** A QBF is either true or false.

**Definition (TQBF)** TQBF := { Set of all TRUE QBF's}

Recall the SAT problem. Given a boolean formula $\phi(\mathbf{x})$ with $n$ free variables $(\mathbf{x})$, we say $\phi(\mathbf{x})$ is satisfiable (or) $\phi(\mathbf{x})$ belongs to SAT iff, there exists a satisfying assignment for the formula $\phi(\mathbf{x})$. An alternate way of defining the SAT problem using QBF would be,

**Definition (SAT)** SAT $= \{\exists x_1 \exists x_2 \cdots \exists x_n \phi(x_1, x_2, \cdots, x_n) \ : \ \phi(x_1, x_2, \cdots, x_n)$ is true $\}$.

## 7.4 TQBF is PSPACE-complete

**Theorem :** TQBF is **PSPACE** $-$ *complete*

**Proof :** To prove TQBF is **PSPACE** $-$ *complete*, we show the following

1. TQBF $\in$ **PSPACE**
2. $L^{'} \leq_p TQBF, \forall \ L^{'} \in$ **PSPACE**

1. TQBF $\in$ **PSPACE** : Consider the QBF $f(\mathbf{x}) = Q_1 x_1 Q_2 x_2 \cdots Q_n x_n \ \ \phi(x_1, x_2, \cdots, x_n)$. Let the size of $\phi(x_1, x_2, \cdots, x_n)$ be $m$. Find $f_{|x_1=0}(\mathbf{x})$ and $f_{|x_1=1}(\mathbf{x})$.

If $Q_1$ is $\exists$ then $f(\mathbf{x}) = f_{|x_1=0}(\mathbf{x}) \vee f_{|x_1=1}(\mathbf{x})$.
If $Q_1$ is $\forall$ then $f(\mathbf{x}) = f_{|x_1=0}(\mathbf{x}) \wedge f_{|x_1=1}(\mathbf{x})$.

The space used to compute $f_{|x_1=0}(\mathbf{x})$ can be reused to compute $f_{|x_1=1}(\mathbf{x})$ after storing the output of $f_{|x_1=0}(\mathbf{x})$. Also we require $O(m)$ space to substiture $x_1 = 0$ and obtain $f_{|x_1=0}(\mathbf{x})$. Therefore we obtain the recursive equation

$$space(n) = space(n-1) + O(m)$$

When $n = 0$ the QBF is a boolean formula of size $O(m)$ with zero variables (only constants). Computing this requires $O(m)$ space. Therefore the total space required is $O(m.n)$. Hence TQBF $\in$ **PSPACE**.

2. $L' \leq_p TQBF, \forall\, L' \in$ **PSPACE** : Let $M$ be a PSPACE machine that decides $L'$. Clearly $M$ uses $m = O(p(n))$ space, where $p(n)$ is a polynomial in $n$. Let $G_{M,\mathbf{x}}$ be the configuration graph corresponding to $(M, \mathbf{x})$. We know, $|G_{M,\mathbf{x}}| = 2^{(O(p(n)))}$.

To show $L' \leq_p$ TQBF we use a polynomial time computable function $\phi(\mathbf{x}) = \psi_{\mathbf{x}}$ such that

$$\mathbf{x} \in L' \iff \phi(\mathbf{x}) \in TQBF$$

For a given input $\mathbf{x}$ we construct a QBF $\psi_{\mathbf{x}}$ such that, $\psi_{\mathbf{x}}$ is true iff the configuration $C_{accept}$ is reachable from $C_{start}$ in $G_{M,\mathbf{x}}$ in at most $2^m$ steps (meaning $M$ accepts $\mathbf{x}$).

To define recursion, we use the notation $\psi^i_{\mathbf{x}}(C_1, C_2)$ to denote the reachability of $C_2$ from $C_1$ in at most $2^i$ steps. The recursion on $i$ is as follows :

a) Base Case : We compute a formula $\psi^0_{\mathbf{x}}(C_1, C_2)$ (is true iff there is an edge from $C_1$ to $C_2$) such that $|\psi^0_{\mathbf{x}}(C_1, C_2)| = O(m^2)$. We can compute this formula by doing local computation as we did in the proof of Cook-Levin theorem (using *Claim* 4.4 in [1]).

b) Induction : By definition $\psi^i_{\mathbf{x}}(C_1, C_2)$ is true iff, there exists a configuration $C_3$ such that there exists paths $C_1$ to $C_3$ and $C_3$ to $C_2$ of length at most $2^{i-1}$. Therefore,

$$\psi^i_{\mathbf{x}}(C_1, C_2) = \exists C_3 \;\; \psi^{i-1}_{\mathbf{x}}(C_1, C_3) \wedge \psi^{i-1}_{\mathbf{x}}(C_3, C_2) \tag{7.1}$$

However, the size of $\psi^i_{\mathbf{x}}$ is twice the size of $\psi^{i-1}_{\mathbf{x}}$. Thus the total size blowup to compute $\psi^m_{\mathbf{x}}(C_{start}, C_{accept})$ would be very high ($O(2^m)$), which is not desirable. Instead we carefully alter the above formula by adding two additional quantifiers such that $\psi_{i-1}$ is used only once instead of twice. It is,

$$\psi^i_{\mathbf{x}}(C_1, C_2) = \exists C_3 \forall D_1 \forall D_2 \; ((D_1 = C_1 \wedge D_2 = C_3) \vee (D_1 = C_3 \wedge D_2 = C_2)) \implies \psi^{i-1}_{\mathbf{x}}(D_1, D_2) \tag{7.2}$$

Equations (7.1) and (7.2) are equivalent. Proof sketch follows : Suppose (7.1) is false. Then for every $C_3$, $\psi^{i-1}_{\mathbf{x}}(D_1, D_2)$ cannot be true for both $(D_1, D_2) = (C_1, C_3)$ and $(D_1, D_2) = (C_3, C_2)$, implying (7.2) is false. Suppose (7.2) is false. This means for every $C_3$, either when $(D_1, D_2) = (C_1, C_3)$ or when $(D_1, D_2) = (C_3, C_2)$, $\psi^{i-1}_{\mathbf{x}}(D_1, D_2)$ evaluates false. This implies (7.1) evaluates false.

We know $\phi_1 \implies \phi_2$ is equivalent to $\neg\phi_1 \vee \phi_2$. Also $\phi_1 = \phi_2$ is equivalent to $(\phi_1 \wedge \phi_2) \vee (\neg\phi_2 \wedge \neg\phi_1)$. Using the above equivalences we can express equation (7.2) in terms of only $\wedge, \vee$ and $\neg$. For example $(D_1 = D_2)$ can be expressed as $((D_1 \wedge D_2) \vee (\neg D_1 \wedge \neg D_2))$.

The size of $\phi_m$ is given by $size(\phi_m) = size(\phi_{m-1}) + O(m) = O(m^2)$. This reduction is polynomial time. In fact this is a log-space reduction as well (We will define log-space reduction in the next lecture). Hence TQBF $\in$ **PSPACE** $-$ *complete*.

## 7.5 Certificate Definition of NL

**Definition (NL) :** A language $L \in$ **NL**, if there is a log-space machine $M$ such that $\mathbf{x} \in L$ iff, $\exists u \in \{0,1\}^{p(|\mathbf{x}|)}$ such that $M(\mathbf{x}, u) = 1$, where $u$ is read-once.

The above definition is equivalent to our previous definition (that is **NL** = **NSPACE**$(O(\log n))$. Let

$NL_1$ be the old definition and $NL_2$ be our new definition. We prove $NL_1 = NL_2$.

i) Let $L \in NL_1$. Then there is a NDTM $N$ that decides $L$ such that $N$ uses $O(\log|\mathbf{x}|)$ space on input $\mathbf{x}$. This implies $L \in NL_2$, because for an input $\mathbf{x}$, $M$ can simulate $N$ by taking $u$ as the sequence of non deterministic choices along an accepting path. Also $M$ is a log-space machine because $N$ decides $\mathbf{x}$ in log-space.

ii) Let $L \in NL_2$. Then there exists a log-space machine $M$ such that $\mathbf{x} \in L$ iff $\exists u \in \{0,1\}^{(p|\mathbf{x}|)}$ such that $M(\mathbf{x}, u) = 1$, where $u$ is read-once. This implies there exists a NDTM $N$ that simulates $M$ as follows: Given input $\mathbf{x}$, $N$ non-deterministically guesses each bit that the verifier $M$ reads from certificate. If $M(\mathbf{x}, u) = 1$ then $N$ reaches accepting state, otherwise $N$ halts and outputs 0. Since $M$ uses log-space (and $u$ is read once), $N$ uses log-space. Hence $NL_1$ is equivalent to $NL_2$.

In this lecture we have studied **PSPACE** – *completeness*. We have also proved that the language TQBF is **PSPACE** – *complete*. The next lecture will be of similar flavour where we study **NL** – *completeness* along with an example (PATH problem). Since **NL** – *completeness* uses log-space reduction, we will also learn about *implicit log space computable functions*.

## References

[1] Sanjeev Arora and Boaz Barak, Computational Complexity: A Modern Approach, *Cambridge University Press*, 2009