University of California, Los Angeles
CS 281 Computability and Complexity

Instructor: Alexander Sherstov
Scribe: Kaixuan Ji
Date: February 12, 2024

LECTURE
10

P-SPACE Completeness and Introduction to Polynomial Hierarchy

In last lectures we have introduced the class of **P-SPACE** and **NP-SPACE**, which are the counterpart of **P** and **NP** regarding to space complexity. Then a natural question is the completeness of these classes: what are the hardest problem in the class in **P-SPACE**? Like the discussion of **NP**-completeness, in this lecture we will discuss the completeness of class **P-SPACE**.

Like the previous lectures, we will first give the definition of the class **P-SPACE**-complete. Then we are going to introduce the concept of quantified boolean formula (QBF), which is the essential for the instantiating of **P-SPACE**-complete problems. Finally, we will give a example of the class **P-SPACE**-complete problem. This example can be seen as an extension of the problem `CNF` with polynomial numbers of quantifiers. Inspired by this, we study the boolean formula with constant numbers of quantifiers and results in a rich class of problems, polynomial hierarchy.

10.1 Definitions

In this section, we will give the definition of several key terms that are necessary for the following part of the lecture. Analogously to **NP**-completeness, we first define **P-SPACE**-hard problems as follows.

Definition 10.1. A language L is **P-SPACE**-hard, if for every $L' \in$ **P-SPACE**, $L' \leq_p L$

Remark 10.2. It worth noticing that in the definition of **P-SPACE**-hard, we requires L' reducible in *polynomial time* to L rather than using polynomial space. Actually, for any language non-trivial (i.e., not empty or universe) $L \in$ **P-SPACE**, we can always reduce L to another non-trivial language L' in polynomial space in the following way. We first pick $y_p \in L'$ and $y_n \notin L'$. Then, given x, we use polynomial space to compute $M(x)$, where M is the TM computing L. We then let $f(x) = y_p$ if $M(x) = 1$ else $f(x) = y_n$, which reduced L to L'. Therefore, we require the reduction to be available in polynomial time.

Now we can give the definition of **P-SPACE**-complete:

Algorithm 1 Evaluate a Quantified Boolean Formula (EVAL)

Require: TQBF $Q_1x_1, Q_2x_2, ..., Q_nx_n, \varphi(x_1, ...x_n)$
Ensure: Value of TQBF
1: Let $v_0 \leftarrow \text{EVAL}(Q_2x_2, ..., Q_nx_n, \varphi(0, x_2, ...x_n))$
2: Let $v_1 \leftarrow \text{EVAL}(Q_2x_2, ..., Q_nx_n, \varphi(1, x_2, ...x_n))$
3: **if** $Q_1 = \forall$ **then**
4: Output $v_0 \wedge v_1$
5: **else**
6: Output $v_0 \vee v_1$
7: **end if**

DEFINITION 10.3. A language L is **P-SPACE**-complete if L is **P-SPACE**-hard and $L \in$ **P-SPACE**.

Before diving into the existence of **P-SPACE**-complete problem, we introduce the following notation:

DEFINITION 10.4. A quantified boolean formula $Q_1x_1, Q_2x_2, ..., Q_nx_n\varphi(x_1, ...x_n)$, where Q_i is one of the two quantifiers $\forall$ and $\exists$, $x_1, ..., x_n$ are boolean variables and φ is a boolean formula of $\{0, 1\}^n$. We further denote the language TQBF as the set of quantified boolean formula that are true.

REMARK 10.5. We would like to stress that a QBF $Q_1x_1, Q_2x_2, ..., Q_nx_n\varphi(x_1, ...x_n)$ is a constant boolean value rather than a function. To be more specific, $\varphi(x_1, ...x_n)$ is a function of $x_1, ..., x_n$, $Q_nx_n\varphi(x_1, ...x_n)$ is a function of variables $x_1, ..., x_{n-1}$ and $Q_2x_2, ..., Q_nx_n\varphi(x_1, ...x_n)$ is a function of x_1.

10.2 The Existence of P-SPACE-complete Problem

With the definition above, we are now prepared to prove the existence of **P-SPACE**-complete problems[1].

THEOREM 10.6. *TQBF is **P-SPACE**-complete.*

Proof. We prove the theorem in two steps. We first prove that TQBF $\in$ **P-SPACE**. Following the proof of **NSPACE**(S(n)) $\subset$ **SPACE**($S(n^2)$), we also consider using recursive algorithms to compute it in a space efficient manner. Here we show the algorithm in Algorithm 1.

Now we analysis the space complexity of Algorithm 1. Since the boolean formula φ accepts n boolean variables as input, the depth of the running stack of Algorithm 1 is n. For each layer in the stack, we only need to compute $v_0 \vee v_1$ or $v_0 \wedge v_1$, as well as storing the call for EVAL($\varphi(0, x_2, ..., x_n)$) and EVAL($\varphi(1, x_2, ..., x_n)$). Therefore, we only need $O(n)$ spaces for each layer of recursive calls. Therefore, the space complexity of Algorithm 1 is $O(n^2)$, which completes our proof.

The next step is to prove that any language $L \in$ **P-SPACE** can be reduced to TQBF in polynomial time. We first consider the Turing Machine (TM) M that compute L in $cS(n)$ spaces, where S is a polynomial. Following the standard approach in space complexity.

Since M is only using $O(S(n))$ space, we can use $m = O(S(n))$ bits to encode each state of M's configuration. Given an specific input x, by the transition rule of M, similar to the proof of Cook-Levine's Theorem, we can construct the boolean formula $\varphi_{M,x}$ for the TM M and input x, such that $\varphi_{M,x}(C, C') = 1$ if and only if configuration C' is a legal succeed of C.

Similar to the previous approaches, we construct the configuration transition (directed) graph G, with vertices C_i and edge from C to C' if and only if $\varphi_{M,x}(C, C') = 1$. Then we know M accepts x is equivalent to there is a path no longer than $2^{cS(|x|)}$ from $C_{M,x}^{\text{START}}$ to $C_{M,x}^{\text{ACCEPT}}$. We denote this claim as $\text{LEADTO}(C_{M,x}^{\text{START}}, C_{M,x}^{\text{ACCEPT}}, 2^{cS(|x|)})$. Our task is to construct a QBF such that it is satisfiable if and only if $\text{LEADTO}(C_{M,x}^{\text{START}}, C_{M,x}^{\text{ACCEPT}}, 2^{cS(|x|)})$ is true.

One naive approach is to follow the proof in Satvich's Theorem. For $k > 0$, we can decompose $\text{LEADTO}(C, C', 2^k)$ in the following way:

$$\text{LEADTO}(C, C', 2^k) = \exists C'',\ \text{LEADTO}(C, C'', 2^{k-1}) \wedge \text{LEADTO}(C'', C', 2^{k-1}), \quad (10.1)$$

where C'' is some legal configuration (which can be represented in boolean form). For the base case $k = 0$, we have:

$$\text{LEADTO}(C, C', 1) = \varphi_{M,x}(C, C').$$

Under this approach, the depth of stack will be $cS(|x|)$. For each recursive step, the size of the boolean formula doubles (i.e., on LEADTO results in two LEADTO). Therefore, the final induced boolean formula will have size $O(2^{cS(|x|)})$, but instead we want the resulting formula have a size that is a polynomial of n.

One deficiency in the naive approach is that in (10.1), we introduce two identical formula $\text{LEADTO}(C, C'', 2^{k-1})$ and $\text{LEADTO}(C'', C', 2^{k-1})$ but only with different variable. Therefore, we can incorporate them and rewrite (10.1) as the follows:

$$\begin{aligned}
&\text{LEADTO}(C, C', 2^k) \\
&= \exists C'',\ \forall A, B,\ [(AB = CC'') \vee (AB = C''C')] \Rightarrow \text{LEADTO}(A, B, 2^{k-1}) \\
&= \exists C'',\ \forall A, B,\ [(AB \neq CC'') \wedge (AB \neq C''C')] \vee \text{LEADTO}(A, B, 2^{k-1}),
\end{aligned}$$

where $\Rightarrow$ means inducing and $=$ means equal, both of which can be represented in standard boolean formula. For the base case $k = 0$, we still have

$$\text{LEADTO}(C, C', 1) = \varphi_{M,x}(C, C').$$

In each recursive layer, the number of quantifiers increases by 3 and the size of boolean formula increase by a constant. Since the depth of recursive stack is $cS(|x|)$, we know that the resulted boolean formula has $O(S(|x|))$ quantifiers and size $O(S(|x|))$. □

REMARK 10.7. We see that in the naive approach, we only introduces $\exists$ as our new quantifiers. Therefore, the final result of this approach is essentially a boolean formula with one quantifier $\exists$, which is actually the language `SAT`. Consequently, if this approach works, we can prove that **P-SPACE** is no harder than **NP.**

Now we have proved the existence of **P-SPACE**-complete problem. We can also rewrite the theorem as the follows:

Algorithm 2 REACHTO

Require: Directed graph $G = (V, E)$ where $|V| = n$, vertices $u, v \in V$
Ensure: Existence of path from u to v

```
t ← n
while t > 0 do
  if u = v then
    Output True
  else
    Guess w
    if (u, w) ∉ E then
      Output No
    else
      u ← w, t ← t − 1
    end if
  end if
end while
```

COROLLARY 10.8. *A language $L \in$ **P-SPACE** if and only if $\exists$ constant c and TM M that runs in polynomial time such that*

$$x \in L \Leftrightarrow \exists u_1, \forall u_2, ... \ M(x, u_1, ...u_{|x|^c})$$

Proof. We first prove the $\Rightarrow$ direction. Given $L \in$ **P-SPACE**, by Theorem 10.6, we have a quantified boolean formula φ takes $x, u_1, ...u_{|x|^c}$ as inputs and

$$x \in L \Leftrightarrow \exists u_1, \forall u_2, ..., \ \varphi(x, u_1, ...u_{|x|^c}).$$

We construct the TM M that compute φ and finish the proof of this statement.

For $\Leftarrow$, we can use an algorithm similar to Algorithm 1. The only difference is to change the boolean formula φ in Algorithm 1 to the Turing machine M. Since space can be reused, the space complexity is $O(|x|^c)$. □

REMARK 10.9. Suppose M is a TM runs in polynomial time, then Corollary 10.8 suggests that adding quantifiers to M expand the expressing ability from **P** to **P-SPACE**. To be succinct, we can say "SPACE=TIME+QUANTIFIERS".

We denote the problem of identifing whether there is a path from one vertex C to another vertex C' in a given graph G as `REACH`. Sawitch's theorem have already shown that `REACH` $\in$ **SPACE**$(\log^2 n)$, we now show that `REACH` $\in$ **NSPACE**$(\log n)$.

THEOREM 10.10. ***REACH*** $\in$ ***NSPACE***$(\log n)$

Proof. We use Algorithm 2 to solve the problem. Since the size of graph is n, we know that if there exists a path from u to v, the length of the shortest path is smaller than n. Therefore, we can simply guess a path of length smaller than n and verify it node by node. Since we need $\log n$ bits to encode the n vertices, the space complexity of Algorithm 2 is $O(\log n)$. □

10.3 Polynomial Hierarchy

In the previous lectures, we have seen the relation between problem classes and boolean formula. We have seen that each language in **P** can be reduced to CNF, and language in **NP** can be reduced to SAT, which is CNF with a single quantifier $\exists$. If we allow polynomial amounts of quantifiers, as suggested by Theorem 10.6, we get **P-SPACE**. Therefore, a natural question is, what if we allow a constant (but a lot more) level of quantifiers?

Another motivation to extend number of quantifiers to constant many is that we are hard to capture some interesting problems with only one quantifiers. For example, the problem of finding the minimal size boolean formula.

$$\texttt{MINIDNF} = \{(\varphi, k) | \exists \text{DNF } |\phi| \leq k, \text{s.t.}, \phi \equiv \varphi\}$$

These suggest that we need a richer class of problems. Now we give the relative definitions.

Definition 10.11. Fix $k \leq 1$, define Σ_k to be the class of languages of L such that $L \in \Sigma_k$ if and only if there exists constant c and TM M runs in polynomial time that

$$x \in L \Leftrightarrow \exists u_1 \in \{0,1\}^{|x|^c}, \forall u_2 \in \{0,1\}^{|x|^c}, ..., M(x, u_1, ..., u_k) = 1$$

Definition 10.12. Fix $k \leq 1$, define Π_k to be the class of languages of L such that $L \in \Pi_k$ if and only if there exists constant c and TM M runs in polynomial time that

$$x \in L \Leftrightarrow \forall u_1 \in \{0,1\}^{|x|^c}, \exists u_2 \in \{0,1\}^{|x|^c}, ..., M(x, u_1, ..., u_k) = 1$$

Remark 10.13. Recall that co$\mathcal{L} = \{L : \bar{L} \in \mathcal{L}\}$, we have $\Pi_k =$coΣ_k.

We first give some properties of these classes.

Proposition 10.14. $\Sigma_0 = \Pi_0 = \boldsymbol{P}$, $\Sigma_1 = \boldsymbol{NP}$

Proposition 10.15. $\Sigma_k \subset \Sigma_{k+1}$, $\Pi_k \subset \Pi_{k+1}$, $\Sigma_k \subset \Pi_{k+1}$, $\Pi_k \subset \Sigma_{k+1}$

The proof of Proposition 10.15 is simple. The first two claims can be verified by simply neglecting the last quantifier in Σ_{k+1} and Π_{k+1}. The last two statements can be verified be neglecting the first quantifier in Σ_{k+1} and Π_{k+1}.

Now we are ready to define the polynomial hierarchy (PH).

Definition 10.16. PH$= \bigcup_{k=0}^{\infty} \Sigma_k = \bigcup_{k=0}^{\infty} \Pi_k$.

We show the structure of PH in Figure

References

[1] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: preliminary report, in5th symp. on theory of computing'. 1973.

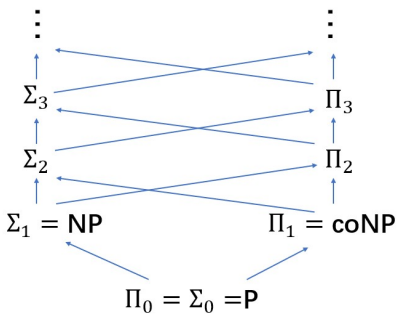

FIGURE 10.1: The structure of polynomial hierarchy. Here the arrows indicate subset. It is conjectured that all the containments are proper.