CS6160 Complexity

Topic: Sipser–Gàcs–Lautemann Theorem, **RL**, **BPL** Date: Oct. 14, 2026
Lecturer: Wei-Kai Lin Scriber: Deebakkarthi Chinnasame Rani

---

1 Student Presentation: Promise Problems [Gol05]

Intuitively, we know that not all problem instances are created equal. Some are hard, some are easy, and some are important. So far, we've forced our Turing machines to work with the **entire** set of bit-strings. What if we want to relax this constraint? What if we only cared about a subset of strings? We capture this idea using *Promise problems*.

We got a glimpse of Promise problems last class with USAT. Let us now define them rigorously.

Definition 1. Promise Problems

A promise problem Π is a pair of non-intersecting sets $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ such that

$$\Pi_{\text{YES}}, \Pi_{\text{NO}} \subseteq \{0,1\}^*$$
$$\Pi_{\text{YES}} \cap \Pi_{\text{NO}} = \varnothing$$
$$\Pi_{\text{YES}} \cup \Pi_{\text{NO}} = \mathsf{promise}$$

Though not part of the definition, one can define a set called $\Pi_{\text{disallowed}}$ which has all the strings we don't care about. Notice that $\Pi_{\text{YES}} \cup \Pi_{\text{NO}} \cup \Pi_{\text{disallowed}} = \{0,1\}^*$.

Using this we can define the promise equivalent for **P**

Definition 2. $\mathcal{P}$

A promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}}) \in \mathcal{P}$, iff $\exists$ a polynomial-time TM A s.t.

$$\forall x \in \Pi_{\text{YES}}, A(x) = 1$$
$$\forall x \in \Pi_{\text{NO}}, A(x) = 0$$

It is easy to see that $\mathbf{P} \subseteq \mathcal{P}$ by having $\Pi_{\text{disallowed}} = \varnothing$

Similarly we can define $\mathcal{NP}, \mathcal{BPP}$.

1.1 Why do we care?

- Special Cases
  - Promise problems are a natural way to capture the essense of special cases.
  - As we saw in last class, USAT – a special case of SAT – is an important problem to study.
  - More generally, we call $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$ a special case of Π' if $\Pi_{\text{YES}} \subseteq \Pi'_{\text{YES}}, \Pi_{\text{NO}} \subseteq \Pi'_{\text{NO}}$
- Hardness of approximation problems

  - Suppose we are satisfied with near-optimal solutions for a problem. How do we capture this? Moreover, how do we study the complexity of finding such a solution?
  - We formulate the task of finding an approximate solution as a promise problem with Π_{YES} having all the solution having a reasonably high optimum value and Π_{NO} containing all the solutions with low optimum value.
  - Example: $\mathsf{gapClique}_{b,s}$ approximates $\mathsf{maxClique}$. Π_{YES} is the set of all the graphs with a clique of size $> b(N)$ and Π_{NO} is the set of all the graphs with no cliques $> s(N)$.

- Complete problems for classes without a complete problem
  - We know that there isn't a **BPP**-complete problem. But there is a $\mathcal{BPP}$-complete problem.
  - Another notable example is the class of statistical zero-knowledge proof systems $\mathcal{SZK}$

Another point that I want to parrot from [Gol05] is that promise problems are infact *more natural* than their non-promise counterparts. The domain of a problem is rarely the entire $\{0,1\}^*$. For instance let us take SAT. We know that SAT only deals with valid boolean formulas. But we usually have the $\{0,1\}^*$ as the domain of SAT. Some of these strings may not represent a valid boolean formula. We overcome this by canonicalizing the input encoding or by considering bad strings to be non-members. Compare this to the promise version of SAT where we can describe Π_{YES} and Π_{NO} as the subset of $\{0,1\}^*$ that are valid boolean formulas. Which seems more natural to you? It is far more elegent to not even consider these instances than to work around them.

1.2 Reductions

Reductions of promise problems face structural challenges. It is clear that any reduction should preserve the Π_{YES} and Π_{NO} instances.

Definition 3. Karp-reduction of Promise problems

A promise problem $\Pi = (\Pi_{\text{YES}}, \Pi_{\text{NO}})$ is *karp-reducible* to $\Pi' = (\Pi'_{\text{YES}}, \Pi'_{\text{NO}})$ if $\exists$ a polynomial time TM f such that

- $x \in \Pi_{\text{YES}} \iff f(x) \in \Pi'_{\text{YES}}$
- $x \in \Pi_{\text{NO}} \iff f(x) \in \Pi'_{\text{NO}}$

We can define Cook and Randomized reductions analogously.

1.3 Example of a structural challenge

Consider the following problem

$$\mathsf{xSAT}(\phi_1, \phi_2) = \begin{cases} \Pi_{\text{YES}}, & \text{if } \phi_1 \in \mathsf{SAT}, \phi_2 \notin \mathsf{SAT} \\ \Pi_{\text{NO}}, & \text{if } \phi_1 \notin \mathsf{SAT}, \phi_2 \in \mathsf{SAT} \end{cases}$$

Theorem 4.

$$\mathcal{NP} \leq_T^P \mathsf{xSAT}$$

Where $\leq_T^P$ is a Cook reduction.

xSAT is then $\mathcal{NP}$-complete under Cook reduction. Notice that the defintion of xSAT requires us to not only to prove membership but also non-membership in SAT which means it is in **co**$\mathcal{NP}$ too. If it is $\mathcal{NP}$-complete and also in $\mathcal{NP} \cap$ **co**$\mathcal{NP}$, does that mean $\mathcal{NP} =$ **co**$\mathcal{NP}$? – **No**. Promise problems under Cook reduction don't have the same transitive property if the target problem is in $\mathcal{NP} \cap$ **co**$\mathcal{NP}$. Without going into much detail, we would require something called a *smart reduction* for this to propagate.

When dealing with promise classes, we have to curb our first instinct to find subsets and supersets. They are not directly comparable to their non-promise counterparts. The reason is very simple – the contents of the sets are different. For example, each element of **P** is a *a set of strings*, whereas each element of $\mathcal{P}$ is *pair of non-intersecting sets*.

Theorem 5. *Sipser–Gàcs–Lautemann theorem [Sip83, Lau83]*

$$\mathbf{BPP} \subseteq \Sigma_2^p \cap \Pi_2^p$$

At first sight there doesn't seem to be anything in randomized computation that would make it related to the polynomial hierarchy. But we'll see how one could transform a randomized computation into a derandomized polynomial hierarchy computation.

Michael Sipser came up with **BPP** $\subseteq$ **PH**, specifically **BPP** $\subseteq \Sigma_4^p \cap \Pi_4^p$. Peter Gàcs tightened the class and proved **BPP** $\subseteq \Sigma_2^p \cap \Pi_2^p$. Clemens Lautemann also proved **BPP** $\subseteq \Sigma_2^p \cap \Pi_2^p$ independently but with a much simpler proof.

Let us remind ourselves that **BPP** is *closed under complement* i.e **BPP** = **coBPP**. We also know that the negation of Σ_2^p is Π_2^p. Then it suffices to just prove **BPP** $\subseteq \Sigma_2^p$.

Proof Sketch. Assume $L \in$ **BPP**. Then we have a probabilistic polynomial time TM M that can compute this with probability greater than $\frac{2}{3}$. Through amplification we can increase this to be very close to 1. Remember that the alternate definition of a PTM is TM with an additional read-only random tape. We denote this as $M(x; r)$. A high probability of success translates to having very few *bad* r-s.

The conversion of **BPP** to **PH** is via creating a *shifter* that would convert the bad r-s to a good one. We say that there exists a shifter for all r such that M always computes L correctly. This looks exactly like a **PH** problem, hence **BPP** $\subseteq \Sigma_2^p$

Proof. If $L \in$ **BPP** $\implies \exists$ poly-time TM M such that

$$\forall x, \Pr_r[M(x; r) = L(x)] \geq 1 - 2^{-2n} \qquad \text{[through amplification]}$$

Let us ponder what this means. The probability is over the random coins r.

$$\begin{aligned} x \in L, \forall r, M(x; r) = 1 &\qquad \text{almost always holds} \\ x \notin L, \neg(\forall r, M(x; r) = 1) &\qquad \text{always holds} \end{aligned}$$

We can derive the following equivalent corollaries: For a given x, most r-s makes M output the right answer. There is no x for which $\forall r$ gives the wrong answer. $\forall x, \exists r$ such that $M(x; r) = L(x)$.

Imagine the worst case scenario. Suppose $M(x; r)$ is wrong. Can we fix it? Since x is fixed, it seems that if we had just gotten a different r we would have outputted the right answer. What if

we had a *fixed* set of shifter that can fix all the wrong instances while not inverting any of the right instances? We describe them as follows:

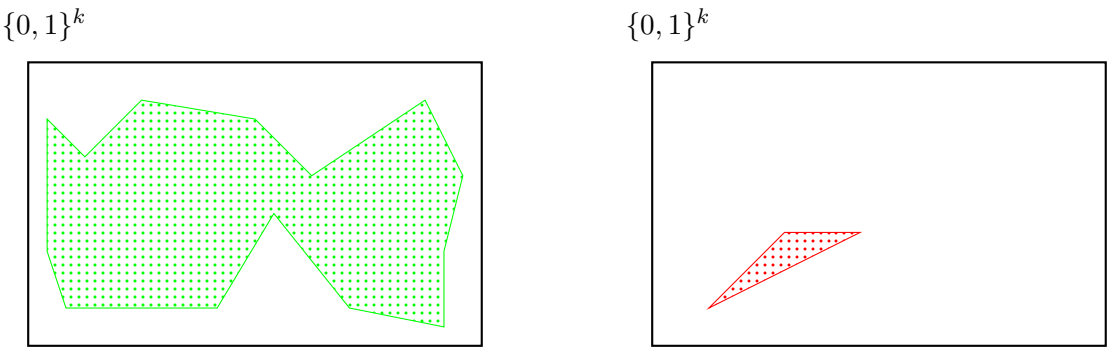
Let $x \in \{0,1\}^n$ and $r \in \{0,1\}^m$ where $m = \mathsf{poly}(n)$. Let us pick $k = 2 \cdot \lceil \frac{m}{n} \rceil$. We formulate the following problem

$$\exists_{u_i \in \{0,1\}^m} u_1 u_2 \dots u_k \forall r \bigvee_{i \in [k]} (M(x; r \oplus u_i) = 1)$$

Restating our two points,

- $\forall x \in L, \exists$ shift $u^* \in \{u_1, u_2, \dots u_k\}$ such that $r \oplus u^*$ makes M output 1
- $\forall x \notin L, \nexists$ shift $u^* \in \{u_1, u_2, \dots u_k\}$ such that $r \oplus u^*$ makes M output 1

Let us fix some x. Let $S_x = \{r \in \{0,1\}^k : M(x; r) = 1\}$. If this $x \in L$ then S_x is large and is our *good* set as seen in 1a. If $x \notin L$ then the set S_x is the *bad* set and is small as seen in 1b. When $x \in L$ the size of the set S_x is *large* – almost as big as $\{0,1\}^k$ itself – whereas when $x \notin L$, S_x is pretty small.



(a) S_x for $x \in L$ (b) S_x for $x \notin L$

Our set of *shifters* $u_1 \dots u_k \in \{0,1\}^k$ tries to move (not move) some r into (out of) our desired region.

Claim 5.1.

$$S \subseteq \{0,1\}^m, |S| \geq (1 - 2^{-2n}) \cdot 2^m$$
$$\implies \exists u_1 \dots u_k \forall r \in \{0,1\}^m \bigvee_{i \in [k]} (r \oplus u_i \in S)$$

Proof. We are going to prove this using the *Probabilistic Method*.

Consider,

$$\Pr_{u_1, \dots, u_k \sim \mathsf{Unif.}} \left[\forall_r \bigvee_i (r \oplus u_i \in S) \right] > 0$$

We need a *single* string u that can convert *any* r into a good one.

Let us just consider a single u for now. Fix some r. What is the probability of not being able to shift this?

$$\Pr_{u \sim \mathsf{Unif.}(\{0,1\}^m)} [r \oplus u \notin S]$$

Since r is fixed and $\oplus$ preserves randomness, we just write this as

$$\begin{aligned}
\Pr_{u \sim \mathsf{Unif.}(\{0,1\}^m)} [u \notin S] &= 1 - \frac{|S|}{|\{0,1\}^m|} && \text{(Size of } S \text{ over size of } \{0,1\}^m) \\
&\leq 1 - \frac{(1 - 2^{-2n})2^m}{2^m} \\
&\leq 2^{-2n}
\end{aligned}$$

This is basically due to the size of S. If S is big enough then picking an $r \oplus u$ outside it is very rare.

As each choice of u is i.i.d w.r.t one another we get

$$\Pr_{\text{uniform } u_1,\ldots,u_k} \left[\bigwedge_i (r \oplus u_i \notin S) \right] < (2^{-2n})^k$$

Substituting the value of k

$$\Pr_{\text{uniform } u_1,\ldots,u_k} \left[\bigwedge_i (r \oplus u_i \notin S) \right] < 2^{-2m}$$

Then over all r

$$\begin{aligned}
\Pr \left[\bigcup_r \left(\bigwedge_i (r \oplus u_i \notin S) \right) \right] &= \sum_r 2^{-2m} \\
&= 2^m \cdot 2^{-2m} \\
&= 2^{-m}
\end{aligned}$$

Which would be basically impossible. Hence we can always find a u such that it can shift any r to a good one. □

Claim 5.2.

$$\begin{aligned}
&S \subseteq \{0,1\}^m, |S| \leq 2^{-2n} \cdot 2^m \\
\implies &\nexists u_1, \ldots, u_k \forall r \bigvee_{i \in [k]} (r \oplus v_i \in S)
\end{aligned}$$

Proof. We see that XOR preserves set size i.e $|S \oplus u_i| = |S|$. This is because either $s_i \oplus u_i \in S$, in which case the set size remains the same, or $s_i \oplus \notin S$, in which case we replace s_i with $s_i \oplus u_i$, which also preserves the size.

If $|S| \leq 2^{m-2n}$ then $\left| \bigcup_i^k S \oplus u_i \right| \leq k \cdot 2^{m-2n} << 2^m$

Hence for any set of $u_1, \ldots, u_k$ we can always an r such that none of them shift r into S. □

Combining both the claims we can now say that $x \in L \iff$

$$\exists u_1, \ldots, u_k \in \{0,1\}^m \forall r \in \{0,1\}^m r \in \bigcup_{i=1}^{k} (S_x + u_i)$$

Notice that there is an $\exists$ followed by a $\forall$ and hence this is in Σ_2^p

□

2 RL, BPL

Definition 6. BPL

A language $L \in$ **BPL** if $\exists$ a PTM M that uses atmost $O(\log(n))$ space and has the following probability

$$\forall x \in \{0,1\}^* \Pr[M(x) = L(x)] \geq \frac{2}{3}$$

Definition 7. RL

A language $L \in$ **RL** if $\exists$ a PTM M that uses atmost $O(\log(n))$ space and has the following probability

$$\forall x \in L, \qquad \Pr[M(x) = L(x)] \geq \frac{2}{3}$$
$$\forall x \notin L, \qquad \Pr[M(x) = L(x)] = 1$$

3 UPATH

We saw that every complexity class has a problem that captures their essense. This is usually (not always) the complete problem for that class. Examples include SAT for **NP**, TQBF for **PSPACE**, etc.

Similarly, UPATH is an important problem for **RL** (We're not gonna discuss its completeness).

Algorithm 1 A **RL** algorithm for UPATH

Require: Undirected graph, source and destination (G, s, t)
Ensure: 1 if s is connected to t, 0 otherwise

```
v ← s
loop n^3 times
    Pick a random vertex u ∈ neighbors(v)
    if v = t then
        return 1
    end if
    v ← u
end loop
return 0
```

The directed version of this is PATH which we saw it to be in **NL**. We don't think PATH can be solved using randomness.

References

[Gol05] Oded Goldreich. On Promise Problems (a survey in memory of Shimon Even [1935-2004]). *Electronic Colloquium on Computational Complexity (ECCC)*, 01 2005.

[Lau83] Clemens Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17(4):215–217, 1983.

[Sip83] Michael Sipser. A complexity theoretic approach to randomness. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing - STOC '83*, STOC '83, page 330–335. ACM Press, 1983.