Foundations of Probabilistic Proofs (Fall 2022)
Note 6: Intro to PCPs
Date: 2022.10.05

This note contains definitions, theorems, facts, *etc.* that are not fully explained in lectures due to limited time. If you think there are anything missing or any mistakes, please contact ziyi.guan@epfl.ch.

1 Complexity classes

In the lecture, we define a PCP system and prove an upper bound $\mathsf{PCP} \subseteq \mathsf{NEXP}$, which leads to the result $\mathsf{PCP}[l, r] \subseteq \mathsf{NTIME}((2^r + l) \cdot poly(c))$. To fully appreciate these results, we need to know the formal definition of these newly mentioned complexity classes.

- $\mathsf{NTIME}(f(n))$: Nondeterministic $f(n)$-Time, where n is the length of the input.
  - Similar to NP, but with $f(n)$-time nondeterministic Turing machines, where $f(\cdot)$ is some constructible function.
  - $\mathsf{DTIME}(f(n))$ is the deterministic version of $\mathsf{NTIME}(f(n))$.
  - $\mathsf{P} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(n^k)$.
  - $\mathsf{NP} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(n^k)$.
- NEXP: Nondeterministic exponential time.
  - $\mathsf{NEXP} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}(2^{n^k})$.
  - $\mathsf{EXP} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}(2^{n^k})$.
  - $\mathsf{NP} \subseteq \mathsf{EXP}$ by going over all possible polynomial length witness.
  - $\mathsf{PSPACE} \subseteq \mathsf{EXP}$: a Turing machine that uses polynomial space can have at most exponential many different configurations by counting.

2 From IP to PCP

In lecture we mention that we have easily convert an interactive protocol to a PCP by putting all proofs with respect to all possible randomness. We give a concrete example of this transformation using the interactive protocol of GNI. In particular, we have that $\mathrm{GNI} \in \mathsf{PCP}(poly(n), 1)$.

Prover $P(G_0, G_1)$	**Verifier** $V(G_0, G_1)$
$\pi = [b_H \in \{0,1\} : b_H = b \text{ where } H \equiv G_b]_{H\text{: graphs with } n \text{ vertices}}$	
	Sample $b \leftarrow^{\$} \{0,1\}$
	Sample random permutation $\phi : [n] \to [n]$
	$H = \phi(G)$
	Accept if $b = \pi[H]$

If $G_0 \not\equiv G_1$, then verifier accept with probability 1. If $G_0 \equiv G_1$, then probability that any π makes the verifier accept is at most $\frac{1}{2}$. This error analysis is the same as the interactive protocol we have.

3 Interactive argument and Kilian's protocol

In the lecture, we mention that PCP can be converted to interactive argument, a relaxation of interactive proof in which we restrict argument prover to be a polynomial-time algorithm. The advantage of the argument prover has over the argument verifier is that the prover gets a private input that enables him to compute the proof within the time constraint. Arguments also have completeness and soundness requirements.

- Completeness: Same as interactive proof.
- Soundness: We have computational soundness. Malicious prover will not be able to falsely convince the verifier with high probability because its computation power is limited to a probabilistic polynomial-time algorithm. This makes interactive arguments easier to achieve than interactive proofs.

Moreover, we are interested in studying how to turn a PCP into an interactive argument, as we already know how to transform an IP to a PCP. In lecture we mention the Kilian's protocol, now we look at it in more details.

3.1 Merkle Tree

Before introducing Kilian's protocol, we need to define formally what is the main tool of this protocol, Merkle tree, as shown in Figure 1.

- A merkle tree is a binary tree.
- The leaves are the input data.
- The internal nodes compute the value of the hash function using its children's values as input.

Note that we use hash functions when computing the values of the nodes in the tree. A good choice of hash functions are collision-resistant hash functions, meaning that a computaionally bounded adversary will have negligible chance to find different inputs x and y to a hash function h such that $h(x) = h(y)$. Therefore, one can show that when using collision-resistant hash functions for a Merkle tree, it is computationally hard to change any value in the tree without changing the root value. This gives us intuition about storing a PCP in a Merkle tree. More specifically, if we treat the proof given by the PCP prover as the input data to a Merkle tree, then the prover could send the root value to the verifier, and the verifier would not have to worry about the prover changing the original proof during the interaction. This intuition leads us to Kilian's protocol.

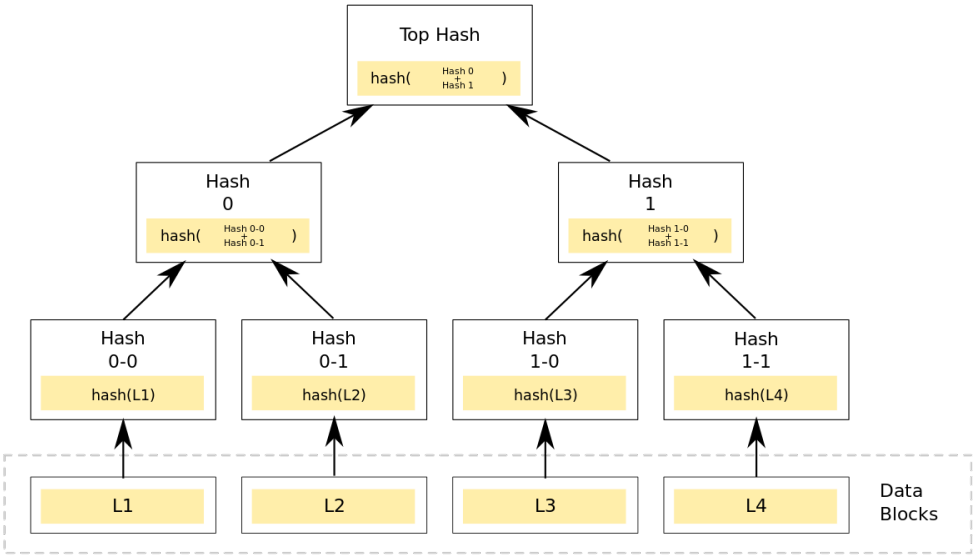

Figure 1: structure of Merkle tree

3.2 Kilian's protocol

The Kilian's protocol shows that any PCP can be combined with Merkle trees to yield argument systems as shown in the table below, assuming that collision-resistant hash functions exist.

Prover $P(x, w)$	**Verifier** $V(x)$
	Send $h \leftarrow^{\$} \mathcal{H}$
$\pi \coloneqq P_{\mathsf{PCP}}(x, w)$	
Compute and send Merkle tree root $\mathrm{MT}_h(\pi)$	
	Sample and send PCP randomness r
Deduce query sets of the PCP verifier	
Send the answers and the paths from answers to root	
	Check the correctness of the path
	Accept if PCP verifier accepts