Computational Complexity: CS41103: Autumn 2021

Instructor: Sudebkumar Prasant Pal

IIT Kharagpur

email: spp@cse.iitkgp.ac.in

October 5, 2022

Preliminary Background

- We know that a single tape Turing machine with one-sided potentially infinite tape, and tape cell alphabet $\{0,1\}$, is equivalent computationally as well as within polynomially bounded complexities to a k-tape Turing machine with arbitrary alphabet K [HU79].
- Apart from 0 and 1, the tape may have the 'blank' and the 'start'/'end' characters.
- What matters is the number of tape cells thus, and not the number of symbols in the tape alphabet.
- The machine will have a finite set S of states, and a transition function δ, deterministic or non-deterministic.
- Formally, a Turing machine is a quadruple $M = (K, S, \delta, s)$. Here K is a finite set of states, $s \in K$ is the initial state, S is a finite set of symbols (alphabet of M).

Preliminary Background (cont.)

- δ maps $K \times S$ to $(K \cup \{h, \text{“}yes\text{''}, \text{“}no\text{''}\}) \times S \times \{l, r, -\}$.
- Theorem 2.1 [Pap94]: Given any k-string Turing machine M operating within time $f(n)$, we can construct a Turing machine M' operating within time $O(f(n)^2)$ and such that, for any input x, $M(x) = M'(x)$.
- One way to do this would be to maintain in the string of M', the “concatenation” of the k strings of M. See [Pap94].
- It is clear that a k^2 term in Theorem 2.1 of [Pap94] will be hidden in the constant factor of the $O(f(n)^2)$ time complexity of M' because we are simply concatenating the k strings !

Preliminary Background

- A smarter way to do the same simulation would be to first list all the first cells of the k tapes, followed by the second k cells and then the third cells, and so on, stated below as in the proof of Claim 1.6 in [AB06].
- A single-tape Turing machine has only one read-write tape which is used for input, work, as well as output. For every function $f : \{0,1\} \rightarrow \{0,1\}$, and a time-constructible function $T : N \rightarrow N$, if f is computable in time $T(n)$ by a TM M using k tapes, then it is computable in time $5kT(n)^2$ by a single-tape TM M'.
- This simulation on a single-tape TM cannot cut down the quadratic growth in the $T(n)$ term though; for this we can view the example of recognizing palindromes and its lower bounds as in Problems 2.8.4 and 2.8.5 in Chapter 2 of [Pap94].

Preliminary Background (cont.)

- However, Problem 2.8.6 of [Pap94] is about cutting down one k factor by representing the k strings, by not concatenating them serially, but instead, by placing them "on top of each other".

Tape compression

- Space and time are the two most important and commonly studied computational resources.
- We study space and time complexities of algorithms on TMs because TMs are very simple machines with finite state control and transition rules, and potentially infinite memory on tape(s).
- By the Church-Turing thesis, all other ‘reasonable’ computational models are equivalent to TMs in computability, as well as, polynomially related in terms of time and space complexities.
- It is expected that using a bigger alphabet set for symbols on the tapes can help in reducing the number of tape cells necessary for computations.
- Definition 1: If for every input string of length n, the TM M scans at most $S(n)$ cells on any writable storage or work tape, then M is said to be an $S(n)$-space bounded TM, or of space complexity $S(n)$.

Tape compression (cont.)

- Definition 2: If for every input word of length n, the TM M makes at most $T(n)$ moves before halting, then M is said to be a $T(n)$-time bounded TM, or of time complexity $T(n)$.
- Tape compression: [Theoreom 12.1 [HU79]]
  It is instructive to first get familiar with tape cell symbol manipulation to the extent that one machine $M2$ is used to simulate another machine $M1$ where each tape cell of $M2$ encodes several (say r) cells of $M1$.
- The purpose of doing such a compression is to use a smaller number of richer cells on the tape by using a more powerful and matching finite-state control and transition function, essentially implying we have better hardware machine.

Tape compression (cont.)

- Using such an encoding $\lceil \frac{S(n)}{r} \rceil$ cells suffice in $M2$ for $S(n)$ cells in $M1$; $\lceil \frac{S(n)}{r} \rceil$ is upper bounded by $\lceil \frac{cS(n)}{2} \rceil \leq cS(n)$, if $rc > 2$, for any choice of $c > 0$.
- Also if $S(n) < r$, then M2 needs just one tape cell. So, M2 can simulate M1 in $cS(n)$ tape cells with compression factor r, where $rc > 2$.
- In other words, we can reduce the space complexity by a factor of c, $c > 0$, by packing every r tape cells of the first machine into a single tape cell of the better machine where $rc > 2$.

Halting space bounded computations

- See Lemma 12.1 in [HU79]. The number of configurations of a $S(n)$-space bounded machine $M1$ is at most $(n+2)sS(n)t^{S(n)}$, where s is the number of states and t is the number of tape symbols.
- So, a simulating TM $M2$ can use a $4st$ base counter with a sufficient number of cells, at least $\log n$ cells but not exceeding $S(n)$ cells, to count the number of moves.
- It is worthwhile working out the details of the simulation where this additional counter cell is appended every time a new tape cell is first visited by the simulating machine $M2$ for this counter.
- In fact, if $M2$ keeps looping having used only i counter cells then the counter will know this when its count reaches $(4st)^{max(i,\log n)}$, which should be at least $(n+2)sS(n)t^{S(n)}$.

Halting space bounded computations (cont.)

- So, we see that languages accepted by space bounded computations are also accepted within the same space bounds by a machine that halts.
- Exercise 1:
  Show that $(4st)^{S(n)}$ exceeds the number of configurations of machine M1 above. All this is about deterministic TMs. What happens if the machines were non-deterministic?

Linear speedup theorem

- Just as we can reduce the number of tape cells by compression, we can achieve similar gains in time complexity as well, using the method of encoding several tape symbols in one tape cell.
- So, in order to speedup by a factor of $c > 0$, we may require to encode some m cells into one cell; we now need to estimate m in terms of c.
- It turns out that as long as $inf_{n\to\infty} \frac{T(n)}{n} = \infty$, choosing m such that $mc > 16$ works out. For details see Theorem 12.3 in [HU79].
- It is also interesting to note that Theorem 7.2 in [HU79] achieves a multitape $T(n)$-time TM's simulation using a single tape TM in $6(T(n))^2$ time. If the multitape machine is apriori speeded up to run in $\frac{T(n)}{\sqrt{6}}$ time using Theorem 12.3 of [HU79], then the simulation by the single tape TM can be done in $(T(n))^2$ time as per Theorem 7.2 in [HU79].

Linear speedup theorem (cont.)

- Such single-tape simulation results hold also for non-deterministic TMs.
- Compare this with the results in Theorem 2.1 of [Pap94] and Claim 1.6 in [AB06].
- A machine M2 can simulate $T(n)$ steps of of a $T(n)$-time machine M1.
- So, M2 requires the following number of moves n (reading the input tape of M1)$+\lceil n/m \rceil$ (encoding m cells of M1 into a single cell of M2)$+8 \times \lceil T(n)/m \rceil$ (simulating $T(n)$ moves of M1 on M2), $\leq n + (n/m + 1) + (8T(n)/m) + 8 \leq n + n/m + 8T(n)/m + 9$.
- If $inf_{n\to\infty} \frac{T(n)}{n} = \infty$, then for any constant d (however large), there is an n_d so that for $n \geq n_d$, we have $T(n)/n \geq d$ or, equivalently, $n \leq T(n)/d$.

Linear speedup theorem (cont.)

- Also, putting $n \geq 9$ (thus $n + 9 \leq 2n$), we have the above upper bound on the number of steps of M2 as $T(n)(2/d + 1/(md) + 8/m)$, for $n \geq n_d$. [These terms come from respectively, $n + 9$, n/m and $8T(n)/m$.]
- Now choosing $m \geq 16/c$ and $d = m/4 + 1/8$, we have the simulation time of M2 at most $cT(n)$.
- Note that $2/d + 1/(md)$ is the same as $8/m$ if $d = m/4 + 1/8$. Also, $8/m$ is less than $c/2$. See [HU79].

Time hierarchy theorem I

- Given more space, we expect to be able to recognize more languages. However, we have seen the linear speed-up and compression theorems; they imply that raising time or space avaiability by merely constant factors will not be helpful.
- In Theorem 12.7 of [HU79], we see that there are recursive languages not in $DTIME(T(n))$ for any total recursive time-bound $T(n)$. Certainly there is a halting TM M that computes $T(n)$.
- Let the ith multitape TM be M_i, whose description is the ith canonical binary string x_i. Define $L = \{x_i | M_i \text{ does not accept } x_i \text{ within } T(|x_i|) \text{ moves}\}$.
- It is not difficult to see that L is indeed recursive.
- We now show that L is not in $DTIME(T(n))$. For the sake of contradiction, we assume that L is in $DTIME(T(n))$.

Time hierarchy theorem I (cont.)

- Let M_i be such that $L = L(M_i)$.
- If x_i is in L then M_i accepts x_i in $T(n)$ steps where $n = |x_i|$. In that case, by the definition of L, we have x_i is not in L, a contradiction.
- If X_i is not in L, then M_i does not accept x_i, and so by the definition of L, we have x_i is in L, a contradiction again.
- So, the assumption that M_i is $T(n)$ time bounded is incorrect. So, L is not in $DTIME(T(n))$.

Space hierarchy theorem I

- Now we see the main result in Theorem 12.8 of [HU79]. What is the encoding of TMs in the proof Theorem 12.8? Why is it that an arbitrary length prefix of 1's is attached to encodings of TMs as defined in Chapter 8?
- How is the simulating machine M forced to use (only) $S2(n)$ space?
- We argue that a UTM M simulates machine M_w on input w in $DSPACE(S2(n))$. It first marks $S2(n)$ cells on the tape. Here, $n = |w|$.
- Why would M_w's tape symbol set cardinality t, determine the amount of space required, $\lceil \log t \rceil$ times times $S1(n)$ for M to simulate M_w on input w? Why is the number k of tapes not playing any role?
- Assume (for the sake of contradiction) that $L(M) = L(M')$ where M' is an $S1(n)$-space bounded TM with t tape symbols.

Space hierarchy theorem I (cont.)

- We show that there is a long string w of such length n that, $\lceil \log t \rceil$ times $S1(n)$ is dominated by $S2(n)$ (shorter strings may not satistfy this $S2(n)$ upper bound), and M_w is M'.
- The simulation by M on input w of TM M_w is such that M accepts w if and only if M_w halts on w rejecting w.
- Observe the way M acts (in terms of accepting or rejecting, in its simulation of M_w on input w); it follows that $L(M)$ is not equal to $L(M_w) = L(M')$.
- In particular, note that $L(M)$ and $L(M_w)$ differ on how they act on w of sufficient length n as required above.
- We conclude therefore that M' being in $DSPACE(S1(n))$ is impossible.

Time hierarchy theorem II [HU79]

- We use two-tape simulations for a sharper bound, with only a logarithmic slowdown.
- We will also assume $T(n)$ is *fully time constructible*. There must be a TM that uses $T(n)$ time on all inputs of length n.
- Let $T2(n)$ be a fully time constructible function and $inf_{n\to\infty}\frac{T1(n)\log T1(n)}{T2(n)} = 0$
- Then we must have a language not in $DTIME(T1(n))$ but in $DTIME(T2(n))$.
- We design a $T2(n)$ time bounded TM M which gets an input w.
- M' treats input w as an encoding of a TM M' and M simulates M' on w.
- The simulation of $T1(n)$ moves of M' by M requires time $cT(n)\log T1(n)$, where c depends on M' but not on $|w|$.

Time hierarchy theorem II [HU79] (cont.)

- After doing $T2(n)$ steps machine M stops simulation and accepts w only if the simulation of M' is completed and M' rejects w.
- Now M' has arbitrarily long encodings. This, if M' is $T1(n)$ time-bounded, there will be a long enough w encoding M' so that $cT1(|w|) \log T1(|w|) \leq T2(|w|)$, so that the simulation can be completed in $T2(n)$ time.
- We observe that w is in $L(M)$ if and only if w is not in $L(M')$. Thus, $L(M) \neq L(M')$ for any M' that is $T1(n)$ time-bounded.

Reachability

- Reachability is a fundamental problem is graphs that tests for connectivity between two specified vertices. There being $n = |V|$ vertices in the graph $G(V, E)$, we need only $\lceil \log n \rceil$ vertices to store the encoding of a vertex.
- BFS and DFS are linear time algorithms but they are not space efficient; sublinear space efficient but time consuming ‘middle first’ search is what we will resort to in establishing Savitch’s theorem [HU79] yielding a deterministic $O((\log n)^2)$-space algorithm.
- We first note that reachability can be solved in nondeterministic $O(\log n)$ space. As long as there is a path Π from a vertex s to a vertex t, there is a valid guess for the next vertex on this path towards t from s.

Reachability (cont.)

- All we need to store is the last guessed move on this path Π by erasing all so far previously visited (guessed) vertices until we make the last guess t, moving from t's previous vertex in Π.
- Clearly, we need no more than $O(\log n)$ space to store store a few vertices. Note that the entire graph can be stored with its edges on a differnt read-only input tape, where we never write anything but can read by moving along the input tape as required.

Savitch's theorem

- Following the notation of [Pap94], we say that predicate $PATH(x, y, i)$ holds if there is a path from x to y in G, of length at most 2^i.
- So, all we need to do is determine whether $PATH(x, y, \lceil \log n \rceil)$ holds.
- If $i = 0$, we can tell whether x and y are connected by an edge or if $x = y$.
- If $i > 1$, then we compute $PATH(x, y, i)$ by testing for each z whether $PATH(x, z, i - 1)$ as well as $PATH(z, y, i - 1)$ hold; any path of length 2^i from x to y must have an intermediate vertex z, such that both x and y are at most 2^{i-1} away from z.
- To realize this idea in a space-efficient manner, we generate all nodes z, one after the other, reusing space.
- A triple $(x, z, i - 1)$ is added to the main work tape and we recurse.

Savitch's theorem (cont.)

- On returning from the recursion if a negative answer to $PATH(x, z, i-1)$ is returned, we erase this triple and move to the next z, rewriting this triple.
- The other case is if a positive answer is returned, we erase the triple $(x, z, i-1)$, and write write $(z, y, i-1)$ after consulting the triple (x, y, i), to the left, to obtain y, and work on deciding whether $PATH(z, y, i-1)$ holds.
- If this is negative, we erase the triple and try the next z; if it is positive, we detect by comparing with triple (x, y, i) to the left that this is the second recursive call, and return a positive answer to $PATH(x, y, i)$.
- So, the work tape acts like a stack of activation records to implement recursion.

Savitch's theorem (cont.)

- Note that the number of triples stored at any moment of time is at most $\lceil \log n \rceil$, the maximum level of recursion.
- Each triple uses at most $3 \log n$ space. So, in total we used $O(\log n)^2$ space.

Immerman-Szelepscenyi theorem

- Following the notation of [Pap94], we would like to count the number of nodes in a graph $G(V, E)$ reachable from a node $s \in V$ using only logarithmic space.
- This would also help us determine the number of nodes not reachable from s within the same space complexity bound.
- Since we wish to use nondeterminism and also restrict to logarithmic space, we must be able to reuse space and operate on only a constant number of $\log n$-sized entities in the work tape.
- Now consider problem of non-deterministically computing the cardinality of the set of vertices reachable from a source vertex s in a directed graph $G(V, E)$ in space proportional to $\log |V|$. Here $|V| = n$.

Immerman-Szelepscenyi theorem (cont.)

- We only elaborate on the subtle steps exploiting space reuse and nondeterminism. Only a constant number of index variables of length $\log n$ are used in the computation.
- Also note that only the “last step” uses nondeterminism, where we determine whether there is a path of length at most $k - 1$ from the node s to a node $v \in V$.
- We define $S(k)$ to be the set of all vertices reachable from s with at most k steps or hops.
- Also $s(k)$ is defined as the number $|S(k)|$ of vertices in $S(k)$. Clearly $s(0) = 1$ and $S(0) = \{s\}$.
- The crucial step is the penultimate level where a conjunction over an iterator is computed for determining whether a certain vertex $u \in V$ is in $S(k)$.

Immerman-Szelepscenyi theorem (cont.)

- This is done by checking for all vertices $v \in V$ whether (i) $v \in S(k-1)$, and whether (ii) $u = v$ or u is directly reachable from v by an edge, that is, whether $G(v, u)$ holds.
- If $u \in S(k)$ then for some v this must hold.
- So, we can write
  for $u = 1, 2, ..., |V|$ {if $u \in S(k)$ then $l \leftarrow l + 1$ };
  $s(k) = l$;
  thereby discovering all $u \in V$ that are in $S(k)$ and computing $s(k)$ as the final value of l above.
- However, if we are trying to determine whether this $v \in S(k-1)$, using only a nondeterminsitic guessing method, then the path of computation making a wrong guess would fail to verify that v is indeed in $S(k-1)$.

Immerman-Szelepscenyi theorem (cont.)

- So, the way we resolve this problem is by running the "for loop" over all v and keeping the count of successes where we actually get certified that a vertex v is indeed discovered to be in $S(k-1)$.
- If this count matches $s(k-1) = |S(k-1)|$, which is already pre-computed and stored in a counter (in the induction process), then we succeed.
- Otherwise, we reject the entire computation in the "for loop".
- The correctness follows from the fact that there is always a correct guessing path for the 'for loop' iterating over each of the vertices $v \in V$.
- function for determining $u \in S(k)$
- $m \leftarrow 0$;
- $answer \leftarrow false$;
- for $v \in V$

Immerman-Szelepscenyi theorem (cont.)

- $\{$if $v \in S(k-1)$ them $m \leftarrow m+1$; if $G(v,u)$ then *answer* $\leftarrow$ *true*; $\}$
- if $m < |S(k-1)|$ then "no" fail, else return *answer*;
- To find if $v \in S(k-1)$ we use $k-1$ guesses w_1 to w_{k-1}, one after another, starting $w_0 = s$ and ending with w_{k-1}, so that for each of the $k-1$ w_p's guessed we have $G(w_{p-1,w_p})$. That is, either $w_{p-1} = w_p$ or there is an edge from w_{p-1} to w_p.
- No wonder this method is called 'inductive counting', using $s(k-1)$, to compute $s(k)$. Now we state all the steps of the full algorithm below.
- Exercises:
- (1) Show that $NSPACE(S(n)) = co - NSPACE(S(n))$ for $S(n) \geq logn$, where $S(n)$ is a fully space constructible function.
- (2) Show that reachability is in $co - NSPACE(\log n)$.

Interactive protocols

- The centralized setting of a computing agent that uses either randomization or simply deterministic computation, is just one end of the spectrum of computational problems.
- If there are multiple agents which are mutually independent but need to co-operate with each other through rules of communication using some protocols, then we may be able to characterize some interesting and non-trivial complexity classes being realized in the languages accepted in such protocols.
- A simple example is the case of Alice and Bob where Alice is all powerful and can use exponential time computations, whereas Bob can only use polynomial time computations.
- In such a scenario, given a boolean formula f in CNF, Bob can ask Alice for a truth asssignment which Alice can always determine as long as the given formula f is satisfiable.

Interactive protocols (cont.)

- However, if the formula f is unsatisfiable then whatever Alice replies to Bob trying to convince Bob that f is satisfiable, Bob will succeed in frustrating such attempts of Alice using polynomial time verification of any claimed assignment to the variables in f.
- This way the protocol captures he class NP.
- A different scenario results if Bob had randomization capabilities based on truly random bits that Bob can generate.
- In that case even if Alice is ignored, Bob can act in such a way that it can accept input strings with bounded error probabilities capturing the power of the class BPP.
- One meaningful notion of realistic computation is enshrined in the class BPP containing all languages L for which there is a nondeterministic polynomially bounded Turing machine N (whose computations are all of the same length) with the following property.

Interactive protocols (cont.)

- For all inputs x, if $x \in L$ then at least $\frac{3}{4}$ of the computations of N on x accept; and if $x \notin L$ then at least $\frac{3}{4}$ of them reject.
- We can tolerate false positives and negatives with an exponentially small error probability.
- Bob can then independently of Alice, decide all languages in BPP by running the BPP algorithm a sufficient number of times and declaring the answer as the majority result.
- A third possibility is that Alice uses its exponential computing capabilities and Bob uses randomization. We now show that this scenario can help running a protocol that captures graph non-isomorphism.

Graph nonisomorphism

- If two graphs given as inputs to Alice by Bob are always isomorphic then Alice will only be getting one kind of information from Bob as long as Bob sends one of these graphs or isomorphs of these graphs to Alice in the interacting steps of a protocol running between them.
- However, if the two graphs given to Alice and Bob as inputs are non-isomorphic then Alice will get some distinguishable information if Bob sends isomorphs or these graphs to Alice in the steps of the protocol.
- So, we can design a protocol accordingly where both are given two graphs G and G'.
- If G and G' are isomorphic then it will be difficult for Alice to convince Bob that the two graphs are non-isomorphic.

Graph nonisomorphism (cont.)

- If the given graphs are non-isomorphic then Alice can always distinguish between two graphs sent by Bob to Alice or even if their isomorphs are sent by Bob to Alice.
- This forms the basis of the folowing protocol for the graph non-isomorphism problem.
- (The graph isomorphism problem is not yet know to be NP-complete but it is in the class NP. Why? The graph non-isomorphism problem is in co-NP therefore. Graph Isomorphism is also not known to be in co-NP, or BPP.)

Graph nonisomorphism

- Bob has to play several rounds. Alice can discriminate if two non-isomorphic graphs are send by Bob in a round.
- However, if Bob sends two isomorphic graphs to Alice, Alice will have no way to distinguish between the two cases (i) both given input graphs are isomorphic, and (ii) the two input graphs are non-isomorphic but Bob decides to send the same graph's isomorphs to Alice as two graphs.
- How will Bob use randomization so that Alice has a small probability of success in fooling/convincing Bob that the two inputs are non-isomorphic even when the two graphs given as input are isomorphic?
- Let us propose that Bob must always send G as one of the graphs to Alice.

Graph nonisomorphism (cont.)

- Then, for the second graph to send to Alice, Bob has the choice of sending an isomorph of G or G'.
- For generating an isomorph, Bob generates a random permutation π. If he chooses to send $\pi(G)$ $(\pi(G'))$ as the second graph then he remembers this by setting a bit b as 1 (0).
- Alice will simply check if the two graphs sent by Bob are isomorphic and send a 1 (0) to Bob if they are checked by Alice to be isomorphic (non-isomorphic).
- This answer of Alice will tally with the bit b of Bob if G and G' are indeed non-isomorphic. So, the round succeeds with probability 1 in this case.
- However, if G and G' are isomprphic, then no matter which graphs Bob sends to Alice, that is, irrespective of the value 1 (0) of the bit b, Alice will always find the graphs sent to her as isomorphic.

Graph nonisomorphism (cont.)

- The only way to convince Bob is to get a correct guess for the 1 (0) value of b, which is private to Bob and not known to Alice.
- This she can guess with only probability of success $\frac{1}{2}$.
- Playing this round $|x|$ times is necessary to reflect the size of the input $x = (G, G')$, whence the probability of success in the protocol is 100% when the two graphs are non-isomorphic, and at most $\frac{1}{2^{|x|}}$, otherwise.
- This shows the power of 'private coins' of the 'verifier' Bob, wheres the tosses are guessed by the 'prover' Alice.

BPP and RP

- Comparing complexity classes, we find that non-determinism and randomization enable capabilities of different kinds.
- Whereas, randomization has relevance in practice in the design of fast algorithms albeit with error probabilities, non-determinism is not realizable in any pragmatic computing model.
- The interest in the relationship between classes like NP and BPP is central and open in complexity theory.
- Surely, NP is very assymetric, with no false positives for non-membership, but with false negatives for membership.
- On the other hand, BPP is symmetric, with both false negatives and false positives.
- It seems BPP and NP are quite incomparable. In fact it is known that $NP \subseteq BPP$ implies $BPP = RP$; asymmetry as in NP enters if $NP \subseteq BPP$.

BPP and RP (cont.)

- The class RP has no false positives but has false negatives. The class $co - RP$ has no false negatives but has false positives.

Probabilistic TMs and BPP

- The class *BPP* can be defined with probability of failure at most $\frac{1}{4}$, or $\frac{1}{3}$, or even any fraction strictly lesser than $\frac{1}{2}$, in deciding membership of strings in its languages. The precise definitions are as follows.
- A *probabilistic Turing machine (PTM)* is a Turing machine with two transition functions.
- To execute a PTM M on an input x, we choose in each step one of the two transition functions with probability $\frac{1}{2}$ each.
- This choice is independent of all previous choices in the computation. We denote by $M(x)$ the random variable corresponding to the $0/1$ value that M writes for output.
- For a function $T : N \to N$, we say that M runs in $T(n)$-time if for any input x, M halts on x within $T(|x|)$ steps in any of the possible random paths.

Probabilistic TMs and BPP (cont.)

- The classes *BPTIME* and *BPP*:
  For $T : N \rightarrow N$ and $L \subseteq \{0,1\}^*$ we say that a PTM M decides L in time $T(n)$ if for every $x \in \{0,1\}^*$, M halts in $T(|x|)$ steps, independently of its random choices, and $Pr[M(x) = L(x)] \geq \frac{2}{3}$.
- We let $BPTIME(T(n))$ be the class of languages decided by PTMs in $O(T(n))$ time and define $BPP = \cup_c BPTIME(n^c)$.
- (BPP, alternative definition) A language L is in BPP if there exists a polynomial-time TM M and a polynomial $p : N \rightarrow N$ such that for every $x \in \{0,1\}^*$, $Pr_{r \in_R}\{0,1\}^{p(|x|)}[M(x,r) = L(x)] \geq \frac{2}{3}$.
- For $c > 0$, let $BPP_{\frac{1}{2}+n^{-c}}$ denote the class of languages L for which there is a polynomial-time PTM M satisfying $Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$ for every $x \in \{0,1\}^*$. Then $BPP_{\frac{1}{2}+n^{-c}} = BPP$.

Probabilistic TMs and BPP (cont.)

- Claim (Probability amplification): Let $L \subseteq \{0,1\}^*$ be a language and suppose that there exists a polynomial-time PTM M such that for every $x \in \{0,1\}^*, Pr[M(x) = L(x)] \geq \frac{1}{2} + |x|^{-c}$. Then for every constant $d > 0$ there exists a polynomial-time PTM M' such that for every $x \in \{0,1\}^*$, $Pr[M'](x) = L(x)] \geq 1 - 2^{-|x|^d}$.
- The machine M' is as follows. For every input $x \in \{0,1\}^*$, M' runs $M(x)$ for $k = 8|x|^{2c+d}$ times obtaining k outputs $y_1, ..., y_k \in \{0,1\}$.
- If the majority of these outputs is 1, then it outputs 1; otherwise, it outputs 0. We use the Chernoff bound to establish the above claim.

Probability amplification

- We discuss the details of tail bounds for *BPP* amplification from [MR00].
- The probability upper bound of the deviation of the sum of n variables (with probability p of being 1), θnp below the mean np, is $e^{-\frac{np\theta^2}{2}}$ (Theorem 4.2 [MR00]).
- So, if $p = \epsilon + \frac{1}{2}$, $\theta = \frac{\epsilon}{\frac{1}{2}+\epsilon}$ then $\theta np = n\epsilon$, whence $(1-\theta)np = np - \theta np = \frac{n}{2}$, the majority vote.
- The tail probability is thus at most $e^{-2np\epsilon^2}$.
- Setting $n = \frac{k}{\epsilon^2}$, we get the bound as e^{-2kp}, where this is e^{-k} as ϵ can be very small.
- So, we can choose any k for the inverse exponential error bound.
- Here, n is the number of times we run the basic *BPP* algorithm on input x. So, n and k are polynomial in $|x|$.

Probability amplification (cont.)

- Now choose $\epsilon = |x|^{-c}$ and $k = |x|^{d}$.

BPP and PH

- The presentation below is from lecture notes of Prof. Jonathan Katz in 2011.
- Say $S \subseteq \{0,1\}^m$ is large if $|S| \geq (1 - \frac{1}{m})2^m$, and small if $|S| < \frac{2^m}{m}$.
- For a string $z \in \{0,1\}^m$ define $S \oplus z = \{s \oplus z | s \in S\}$.
- If S is small, then for all $z_1, ..., z_m \in \{0,1\}^m$ we have $\cup_i (S \oplus z_i) \neq \{0,1\}^m$.
- Otherwise (that is, when S is big), there exists $z_1, ..., z_m \in \{0,1\}^m$ such that $\cup_i (S \oplus z_i) = \{0,1\}^m$.
- This is because $| \cup_i (S \oplus z_i)| \leq \Sigma_i |(S \oplus z_i)| = m.|S| < 2^m$, when S is small.
- Here we had m strings of m bits each. So we see that m translates of S cannot cover the whole space when S is small, and when S is big, some set of m translates of S can cover the space.

BPP and PH (cont.)

- This later part is shown using a probabilistic argument[1].
- Note here that the space is of m-bit vectors and also the number of strings is m.
- We now use these results as follows.
- Given $L \in BPP$, there exist a polynomial m and an algorithm M such that M uses $m(|x|)$ random strings and errs with probability less than $1/m$.
- For any input x, let $S_x \subseteq \{0,1\}^{m(|x|)}$ denote the set of random strings for which $M(x;r)$ outputs 1.
- Thus, if $x \in L$, taking $m = m(|x|))$ we have $|S_x| > (1-\frac{1}{m})^{2^m}$, while if $x \notin L$ then $|S_x| < \frac{2^m}{m}$.
- This leads to the following Σ_2 characterization of L:
- $x \in L$ if and only if $\exists z_1, ..., z_m \in \{0,1\}^m \forall y \{0,1\}^m y \in \cup_i (S_x \oplus z_i)$.
- The condition $y \in \cup_i (S_x \oplus z_i)$ can be efficiently verified by checking if $M(x; y \oplus z_i) = 1$ for some i.

[1] Consider the probability that some specific y is not in $\cup_i (S \oplus z_i)$. This is given by: $Pr_{z_1,\ldots,z_m \in \{0,1\}^m}[y \notin \cup_i (S \oplus z_i)] = \Pi_i Pr_{z \in \{0,1\}^m}[y \notin (S \oplus z)] \leq (\frac{1}{m})^m$. The probability that some y is not covered is at most 2^m times, that is, $(\frac{2}{m})^m$. So, there is an

BPP and Circuits

- We will use the probabilistic method, whereby we show only the existence of a circuit family for any language $L \in BPP$.
- Let N be the probabilistc TM that accepts the BPP language L in $p(n)$ time, where n is the input size.
- For each $x \in \{O, I\}^n$, at most one quarter of the computations are bad for deciding the language $L \in BPP$ for input x of length n.
- Consider a sequence of bit strings $A_n = (a_1, ..., a_m)$ with $a_i \in \{O, 1\}^{p(n)}$ for $i = 1, ..., m$, where $p(n)$ is the length of the computations of N on inputs of length n, and $m = 12(n+1)$.
- The circuit C_n, on input x, simulates N with each string of choices in the sequence A_n, and then takes the majority of the m outcomes for declaring its output.
- So, C_n has a polynomial number of gates.

BPP and Circuits (cont.)

- By a probablistic argument, we must show that there is a sequence A_n such that C_n works correctly, for every input x of size n !
- So, we will show that for all $n > 0$ there is a sequence A_n of $m = 12(n+1)$ bit strings such that for all inputs x with $|x| = n$, fewer than half of the choices in that sequence A_n are bad strings.
- We do not care about which of the m strings are bad for any x but we know that the same sequence A_n works for every x.
- Since the strings in A_n were picked randomly and independently, the expected number of *bad* strings is at most $\frac{m}{4}$ for input x.
- By the Chernoff bound (Lemma 11.9), the probability that the number of bad bit strings in the randomly chosen A_n is $\frac{m}{2}$ or more, is at most $\frac{1}{2^{n+1}}^2$.
- This inequality holds for each $x \in \{O, 1\}^n$, separately.

BPP and Circuits (cont.)

- So, the probability that there is an x with no accepting sequence A_n for C_n is at most the sum of these probabilities among all $x \in \{O, 1\}^n$, and this sum probability is at most $\frac{1}{2}$.

[2]Lemma 11.9 of [Pap94] uses $\theta = 1, p = 0.25$ and n as m to get probability less than $e^{-\frac{m}{12}} < \frac{1}{2^{n+1}}$.

BPP and Circuits

- So, at most half of the possible $2^{p(n)(12(n+1))}$ A_n sequences of $p(|x|)$-length strings are bad for some x of length n.
- Subtracting from the total number of possible A_n sequences of m $p(|x|)$-length strings, we can conclude that at least half of the elements of this space are selections that have an accepting choice for each x in C_n.
- Which ones are these we do not know.

Simpler circuit

- Suppose $L \in BPP$. By the alternative definition of BPP we can reduce the error to show a TM M that on any input of size n needs m random bits and such that for every $x \in \{0,1\}^n$, $Pr_r[M(x,r) \neq L(x)] \leq 2^{n-1}$.
- A string $r \in \{0,1\}^m$ is called *bad* for an input $x \in \{0,1\}^n$ if $M(x,r) \neq L(x)$ and, *good* otherwise.
- For a given x, at most $\frac{2^m}{2^{n+1}}$ strings r are bad for x.
- Adding over all $x \in \{0,1\}^n$, there are at most $2^n \frac{2^m}{2^{n+1}} = \frac{2^m}{2}$ strings r that are bad for some x.
- Therefore there is at least one string $r_0 \in \{0,1\}^m$ that is good for every $x \in \{0,1\}^n$.
- We can burn in string r_0 in a circuit C that on input x outputs $M(x, r_0)$.
- The circuit C will satisfy $C(x) = L(x)$ for every $x \in \{0,1\}^n$.

Circuits

- Polynomial sized Boolean circuits may have logarithmic or even polynomial depth.
- We will look at some basic circuit characterizations of important complexity classes, including those for parallel computation as circuits present a natural model for parallel computing.
- Caution is used while defining infinite families of circuits so that the accepted languages may remain decidable (see Proposition 11.2 [Pap94], for an undecidable language with a polynomial family of circuits)[3].
- To avoid this we have the following notion. So, we carefully define what we call *uniformly polynomial circuits* as follows.
- A *family* $C = (C_0, C_1, \ldots)$ *of circuits* is said to be *uniform* if there is a $\log n$-space bounded Turing machine N which on input 1^n outputs C_n.

Circuits (cont.)

- We say that a language L has *uniformly polynomial circuits* if there is a uniform family of polynomial circuits $(C_0, C_1, \ldots)$ that decides L.
- It turns out that the class P is precisely the class of uniformly polynomial circuits (see Theorem 11.4 in [Pap94]).
- We will also see that all languages in BPP too have polynomial circuits, though may not be polynomially uniform citcuits ! (Theorem 11.6 of [Pap94]).
- It remains open whether $BPP = P$.
- We note that some (many) Boolean functions may need large circuits.
- In Theorem 4.3 of [Pap94] (by Shannon in 1949), we see that an n-ary Boolean function can be computed only by circuits of more than $\frac{2^n}{2n}$ gates.
- How many circuits can we have for n-ary Boolean function computations with m gates?

Circuits (cont.)

- We can have a gate of $n+5$ types, AND, OR, NOT, 0 and 1, and the n inputs. Since there are m gates, each of the at most two inputs can be from the outputs of at most m^2 combinations of gates, giving a very gross upper bound of m gates with at most $(n+5)m^2$ possibilities.
- So, for m gates we have no more than $((n+5)m^2)^m$ possibilities computing one of 2^{2^n} possible Boolean functions.
- Putting $m = \frac{2^n}{2n}$, we see that we have more functions than circuits.
- So, clearly just $m = \frac{2^n}{2n}$ gates will not suffice !
- However, we know that languages in P have polynomial (sized) circuits, by virtue of the proof of the P-completeness of CVP using a logarithmic space reduction (see Theorem 8.1 of [Pap94]).
- Note that every Boolean expression has a CNF and a DNF representation.

Circuits (cont.)

- It is also interesting that unsatisfiable expressions are difficult to construct since they must be false under all truth assignments (see Example 4.2 in [Pap94]).
- Exercises for practice: Problems 4.4.5, 4.4.8, 4.4.10, 4.4.12 and 15.5.4.
- Theorem 15.1: If $L \subseteq \{0,1\}^*$ is in $PT/WK(f(n), g(n))$, then there is a uniform PRAM that computes the corresponding function F_L mapping $\{O,1\}^*$ to $\{O,1\}$ in parallel time $O(f(n))$ using $O(\frac{g(n)}{f(n)})$ processors.

Circuits (cont.)

- Theorem 15.2: Suppose that a function F can be computed by a uniform PRAM in parallel time $f(n)$ with $g(n)$ processors, where $f(n)$ and $g(n)$ can be computed from 1^n in logarithmic space. Then there is a uniform family of circuits of depth $O(f(n)(f(n) + logn))$ and size $O(g(n)f(n)(n^k f(n) + g(n)))$ which computes the binary representation of F, where n^k is the time bound of the logarithmic space Turing machine which on input 1^n outputs the nth PRAM in the family.
- Let $NC = PT/WK(\log^k n, n^k)$ to be the class of all problems solvable in polylogarithmic parallel time with polynomial amount of total work.
- We now observe that $NC_1 \subseteq L \subseteq NL \subseteq NC_2$.
- We show that the non-trivial first and third inclusions hold; the second one is trivial.

Circuits (cont.)

- For showing the first inclusion we cascade a set of three logspace transformations so that the whole process is also a logspace transformation.
- The first one is the building of the circuit for the NC_1 language by its characterization as a uniform family of circuits. The output of this step is the description of all the gates of the constructed circuit.
- In a circuit, a gate may have outdegree more than one; this is the sharing of common subexpressions, possible in circuits.
- The second logarithmic space-bounded algorithm takes this circuit and transforms it into an equivalent circuit with all out degrees one.
- This is done by considering all possible paths in the original circuit. Trace the paths starting from the output and reaching the inputs.

Circuits (cont.)

- A path is not represented by the names of the gates in the path. Instead an economical bit string of length equal to that of the path is used where each bit indicates whether the next gate visited in the path is the first or the second predecessor of the previous gate.
- For the single branch out of a NOT gate we use 0.
- Since the given circuit has logarithmic depth, these paths have logarithmic length representations.
- The output gate will be labeled as ϵ, the empty-string.
- Its first predecessor will be labeled 0, its second 1, the first predecessor of 1 will be labeled 10, and so on.
- Gates reachable by several paths will are naturally represented many times, once for every path that reaches them.
- The gates and the connections in this new circuit can be generated one-by-one, reusing space.

Circuits (cont.)

- In the end we have an equivalent tree-like circuit whose gates are labeled by bit strings of logarithmic length.
- So, the new circuit is a list of bit strings.
- The third algorithm evaluates the output gate of the tree-like circuit.
- To evaluate an AND gate labeled by the string g, the algorithm recursively evaluates its first predecessor $g0$.
- If the first predecessor's value is true, then we must also evaluate the second predecessor, $g1$.
- For OR gates the roles of true and false are reversed.
- For NOT gates we simply evaluate the unique input and return the opposite value, and in the case of true or false gates there is nothing to do.

Circuits (cont.)

- Once the evaluation of a gate is finished, the evaluation of its successor (the unique gate to which it is a predecessor, recall that we are evaluating a tree-like circuit) is resumed.
- The label of the successor can be obtained by simply omitting the last bit of the current label.
- When we finish the evaluation of the output, we have the value of the circuit and we are done.

[3] Proposition 11.2 of [Pap94] shows an example of an unrealistic model of computation where we are allowed to use unbounded amounts of computation to construct each circuit in the family.

PH

- Recall the definition of the class NP in Proposition 9.1 of [Pap94].
- See that this is generalized for defining/characterizing PH in Corollary 2 in Chapter 17 of [Pap94]; the equivalent definition for PH is seen in Definition 5.3 in [AB06], following the Definition 5.1 of Σ_2^p in [AB06].
- Note that as per the above definitions, $\Sigma_1^p = NP$.
- For every i, we define $\Pi_i^p = co - \Sigma_1^p = \{L^c | L \in \Sigma_1^p\}$, where L^c is the complement language $\Sigma^* \setminus L$ of L.

PH

- So, $\Pi_1^p = co - NP$. $\Sigma_2^p = NP^{SAT} = NP^{NP}$, since SAT is NP-complete. $\Pi_2^p = co - NP^{SAT} = co - NP^{NP}$.
- Later, in Theorem 5.12 of Section 5.5 of [AB06], the characterization of the polynomial hierarchy in presented via oracles machines, as in Definition 17.2[4] of Section 17.2 in [Pap94].

[4]The polynomial hierarchy is the following sequence of classes: $\Delta_0^P = \Sigma_0^P = \Pi_0^P = P$; and for all $i > 0$, $\Delta_{i+1}^P = P^{\Sigma_i^P}$, $\Sigma_{i+1}^P = NP^{\Sigma_i^P}$, $\Pi_{i+1}^P = co - NP^{\Sigma_i^P}$, and $PH = \cup_i \Sigma_i^P$.

PH

- Note that although we can simulate all the non-determinism / oracles in the definition of PH within $PSPACE$, we do not know whether $PH = PSPACE$.
- Note that for every i, $\Sigma_i^P \subseteq \Pi_{i+1}^P \subseteq \Sigma_{i+2}^P$.
- Thus, $PH = \cup_{i>0}\Sigma_i^P = \cup_{i>0}\Pi_i$. Try problems 5.1, 5.3, 5.10, 5.12 from [AB06].

PH

- Suppose that the cities in a Euclidean instance of the TSP are the vertices of a convex polygon.
- Then not only is the optimum tour easy to find (it is the perimeter of the polygon) but the instance has the master tour property: There is a tour such that the optimum tour of any subset of cities is obtained by simply omitting from the master tour the cities not in the subset.
- Problem 1: Show that deciding whether a given instance of the TSP has the master tour property is in the second level of PH, in Σ_2^P (Problem 17.3.11 [Pap94]).

PH

- Firstly, we can guess an optimal TSP solution/tour/permutation Π for the full instance.
- Then for every subset S of cities, and for every (sub)permutation of S, we must verify that none of these permutations of S has tour cost lesser than that of the permutation Π_S subsumed by Π on S.
- We know that converting Boolean expressions in disjunctive normal form to conjunctive normal form can be exponential in the worst case, simply because the output may be exponentially long in the input.
- But suppose the output is small. In particular, consider the following problem: We are given a Boolean expression in disjunctive normal form, and an integer B.

PH

- We are asked whether the conjunctive normal form has B or fewer clauses.
- Problem 2: [Problem 17.3.12 in [Pap94]] Show that the problem of determining whether a CNF has fewer clauses than the given DNF is in the second level of PH. Hint: Show that it is in Σ_2^P.
- Guess a CNF formula of B or fewer clauses and check for all inputs whether the DNF agrees with the guessed CNF.
- Now consider the problem: MINIMUM CIRCUIT: Given a Boolean circuit C, is it true that there is no circuit with fewer gates that computes the same Boolean function?

PH

- Problem 3: Show that MINIMUM CIRCUIT is in Π_2^P. For hints, see [Pap94].
- Whatever be a circuit C' with fewer gates, we must have an input x so that $C'(x) \neq C(x)$.
- So, universal quantification for C' would choose all such possibilities and for each such possibility a guessed certificate x will be used for verifying in polynomial time that the two circuits differ on outcomes for x.

PH

- Problem 4: In the SUCCINCT SET COVER problem [AB06], we are given a collection $S = \{\phi_1, \phi_2, ..., \phi_m\}$ of 3-DNF formulas on n variables, and an integer k.
- We need to find whether there is a subset $S' \subseteq \{1, 2, ..., m\}$ of at most k elements for which $\vee_{i \in S'} \phi_i$ evaluates to 1 for every assignment to the variables.
- SUCCINCT SET COVER is thus in Σ_2^P.

PH

- Problem 5: The Vapnik-Chervonenkis (VC) dimension: If $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ is a collection of subsets of a finite set U, the VC dimension of $\mathcal{S}$, denoted $VC(\mathcal{S})$, is the size of the largest set $X \subseteq U$ such that for every $X' \subseteq X$, there is an i for which $S_i \cap X = X'$.
- We say that X is *shattered* by $\mathcal{S}$; every non-empty subset of X is the *projection* of X in some set $S_i \in \mathcal{S}$.
- A Boolean circuit C succinctly represents collection $\mathcal{S}$ if S_i consists of exactly those elements $x \in U$ for which $C(i, x) = 1$.
- Now let $VC - DIMENSION = \{< C, k >$: C represents a collection $\mathcal{S}$ such that $VC(\mathcal{S}) \geq k\}$.
- Show that $VC - DIMENSION \in \Sigma_3^P$. Hint: Guess X and for all $X' \subseteq X$ see whether for some S_i, $X' = X \cup S_i$.

EXPCOM

- Let *EXPCOM* be the following language $\{< M, x, 1^n > : M$ outputs 1 on input x within 2^n steps$\}$.
- Then $P^{EXPCOM} = NP^{EXPCOM} = EXP = \cup_c DTIME(2^{n^c})$ [AB06].
- An oracle to *EXPCOM* allows one to perform an exponential-time computation in one call, so that $EXP \subseteq P^{EXPCOM}$.
- To see this, let $L \in EXP$ be decided by a DTM M_L running in time n^c, $c > 0$.
- A DTM running in polynomial time can pass on $< M_L, x, 1^{|x|^c} >$ as a query for *EXPCOM* so as to decide the behaviour of M_L on x in $2^{|x|^c}$ time.
- So, $L \in P^{EXPCOM}$. Thus $EXP \subseteq P^{EXPCOM}$.
- It is trivial that $P^{EXPCOM} \subseteq NP^{EXPCOM}$.

EXPCOM

- On the other hand, if M is a nondeterministic polynomial-time oracle TM, we can simulate its execution with a *EXPCOM* oracle, in exponential time.
- Exponential time suffices both to enumerate all of M's nondeterministic choices, and also to answer the *EXPCOM* oracle queries.
- To see this let $L \in NP^{EXPCOM}$.
- So, a polynomial length guess is provided as a query to *EXPCOM* by an NDTM M.
- However, an *EXP* machine can not only try out all guesses of M in exponential time but also answer all the queries made by M in exponential time.
- So, $L \in EXP$, forcing $NP^{EXPCOM} \subseteq EXP$.
- Thus $EXP \subseteq P^{EXPCOM} \subseteq NP^{EXPCOM} \subseteq EXP$.

PSPACE oracle

- There is an oracle A for which $P^A = NP^A$ [Pap94].
- Take A to be any PSPACE-complete language.
- We have $PSPACE \subseteq P^A \subseteq NP^A \subseteq NPSPACE \subseteq PSPACE$. Hence, $P^A = NP^A$.
- The second inclusion above is trivial and the fourth inclusion is due to Savitch's theorem.
- For the first one, let L be any $PSPACE$ language.
- The language L can be decided by a polynomial-time deterministic Turing machine that performs the reduction from L to A in polynomial time, and then uses the oracle for A only once.
- For the third inclusion, any nondeterministic polynomial-time Turing machine with oracle A can be simulated by a nondeterministic polynomial space-bounded Turing machine, which resolves the queries to A by itself, in polynomial space.

Karp-Lipton theorem

- We now prove the Karp-Lipton theorem.
- We show that if $NP \subseteq P/poly$ then $\Pi_2^P \subseteq \Sigma_2^P$.
- We know that this implies $\forall k \geq 2$, $\Sigma_k^P = \Sigma_2^P$.
- Let $L \in \Pi_2^P$. So, there is a polynomial $p()$ and a polynomial-time computable $F()$ such that $x \in L$ iff $\forall y1, |y1| \leq p(|x|) \exists y2, |y2| \leq p(|x|), F(x, y1, y2) = 1$.
- This is by the definition/characterization of Π_2^P.

Karp-Lipton theorem contd.

- We can show that, for every n, there is a circuit C_n of size polynomial in n such that for every x of length n and every $y1$, $|y1| \leq p(|x|)$, we have
  $\exists y2, |y2| \leq p(|x|) F(x, y1, y2) = 1$ if and only if
  $F(x, y1, C_n(x, y1)) = 1$,
  due to the premise of the Karp-Lipton theorem that $NP \in P/poly$[5].
- (Here, $y1$ is inherited as an argument in $F()$ by definition and $y2$ is the second existential guess string captured within the NP characterization as a certificate, which is in turn realized by the circuit C_n for $F()$ with arguments x, $y1$ as $y2 = C_n(x, y1)$.)

[5]This needs some elaboration which we skip here.

Karp-Lipton theorem contd.

- Let $q(n)$ be a polynomial upper bound on the size of C_n.
- So, for inputs x of length n, we can write
  $x \in L$ iff $\exists C, |C| \leq q(n), \forall y1, |y1| \leq p(n), F(x, y1, C(x, y1)) = 1$, showing that L is in Σ_2^P.

Meyer's theorem

- If $EXP \subseteq P/poly$ then $EXP = \Sigma_2^P$.
- For any $L \in EXP$, L is decided in $2^{p(n)}$ time by an *oblivious* TM M, where p is some polynomial.
- We show that $L \in \Sigma_2^P$ by showing that $\exists C \in \{0,1\}^{q(n)} \forall i, i_1, ..., i_k \in \{0,1\}^{p(n)} T(x, C(i), C(i_1), ..., C(i_k)) = 1$.
- Here, C is a polynomial $q(n)$-size described circuit and $T()$ is a polynomial tiem computable function with $k+2$ arguments.
- We use the *oblivious* TM convention of Claim 1.6 and Remark 1.7, as used nicely in Theorem 6.6 and elsewhere in [AB06] for simulating k-tape TMs based on local *snapsshots* at each step of the TM in the vicinity of the head positions on the tapes.

Meyer's theorem contd.

- Let $x \in \{0,1\}^n$ be some input string. For every $i \in [2^{p(n)}]$, we denote by z_i the encoding of the ith snapshot of Ms execution on input x as in the proof of Theorem 6.6 [AB06].
  Since M has k tapes, we have
  $x \in L$ iff for every $k+1$ indices $i, i_1, \ldots, i_k$, the snapshots $z_i, z_{i_1}, \ldots, z_{i_k}$ satisfy some easily checkable criteria.
- If z_i is the last snapshot, then it should encode outputting 1, and if $i_1, \ldots, i_k$ are the last indices where Ms heads were in the same locations as in i, then the values read in z_i should be consistent with the input and the values written in $z_{i_1}, \ldots, z_{i_k}$.
- (These indices can be represented in polynomial time.)

Meyer's theorem contd.

- Since $EXP \subseteq P/poly$ by assumption in the premise of this theorem, there is a $q(n)$-sized circuit C, for some polynomial q that computes z_i from i.
- Now the main point is that the correctness of the transcript implicitly computed by this circuit can be expressed as a co-NP predicate that checks whether the transcript satisfies all local criteria over all steps of the computation.
- Hence, $x \in L$ iff the following condition is true $\exists C \in \{0,1\}^{q(n)} \forall i, i_1, ..., i_k \in \{0,1\}^{p(n)} T(x, C(i), C(i_1), ..., C(i_k)) = 1$.

Computing with advice.

- Let $T, a : N \to N$ be functions.
- The class of languages decidable by time-$T(n)$ TMs with $a(n)$ bits of advice, denoted $DTIME(T(n))/a(n)$, contains every L such that there exists a sequence $\{\alpha_n\}n \in N$ of strings with
- $\alpha_n \in \{0,1\}^{a(n)}$ and a TM M satisfying
- $M(x, \alpha_n) = 1$ iff $x \in L$, for every $x \in \{0,1\}^n$, where on input (x, α_n) the machine M runs for at most $O(T(n))$ steps.
- Every unary language can be be decided by a polynomial time Turing machine with 1 bit of advice. The advice string for inputs of length n is the single bit indicating whether or not 1^n is in the language.

Polynomial advice.

- If L$\in$ $P/poly$, then it is computable by a polynomial-sized circuit family $\{C_n\}$.
- We use the description of C_n as an *advice* string on inputs of size n, where the TM is simply the polynomial-time TM M that on input a string x and a string representing an n-input circuit C outputs $C(x)$.
- Conversely, if L is decidable by a polynomial-time Turing machine M with access to an advice family $\{\alpha_n\}_{n\in N}$ in N of size $a(n)$ for some polynomial a, then we can use the construction of Theorem 6.6 to construct for every n a polynomial-sized circuit D_n such that on every $x \in \{0,1\}^n$, $\alpha \in \{0,1\}^{a(n)}$, $D_n(x,\alpha) = M(x,\alpha)$.

Polynomial advice.

- We let the circuit C_n be the polynomial circuit that given x computes the value $D_n(x, \alpha_n)$.
- That is, C_n is equal to the circuit D_n with the string α_n hard-wired as its second input.

References

S. Arora and B. Barak, "Computational complexity: A modern approach,", 2006. [Online]. Available: http://theory.cs.princeton.edu/complexity/.

J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

R. Motwani and P. Raghavan, *Randomized Algorithms*. Cambridge, 2000.

C. H. Papadimitriou, *Computational complexity.* Addison-Wesley, 1994, pp. I–XV, 1–523, ISBN: 978-0-201-53082-7.