Complexity Theory
Boolean Circuits Prep

1. Some things / notes to remember i think

Solution:

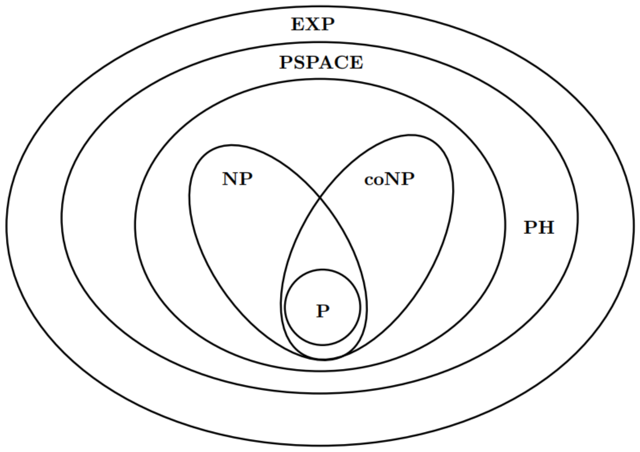
1. $P \subseteq P/Poly$
2. $P/Poly \subseteq P$?

   No, since $P/Poly$ can decide undecidable languages. For example, the Halting Problem in unary.
3. Every unary language $L \subseteq \{1\}^*$ is in $P/Poly$.

   Each circuit C_n is just a constant indicating whether 1^n is in L.

For every space constructible $S : \mathbb{N} \to \mathbb{N}$, $\mathbf{DTIME}(S(n)) \subseteq \mathbf{SPACE}(S(n)) \subseteq \mathbf{NSPACE}(S(n)) \subseteq \mathbf{DTIME}(2^{O(S(n))})$.

So we have: $\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PH} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXP}$, as shown in Figure 1.



Theorem 10. *A language $L \in \mathbf{P}$ if and only if there exists a logspace uniform circuit family computing L. (That is, there exists a logspace TM that on input 1^n outputs the description of the n'th circuit in the family.)*

Proof. The "only if" direction follows immediately from Theorem 6. For the "if" direction, suppose L is decidable by a circuit family generated by logspace TM N. Then the following poly-time TM decides L:
On input x:

1. Generate circuit $C_n = N(1^{|x|})$
2. Evaluate $C_n(x)$.

Logspace uniformity guarantees that C_n takes logspace, and hence polynomial time, to construct. Moreover, since the circuit has polynomial size, it takes polynomial time to evaluate. $\square$

2. Describe a decidable language that is in P/Poly but not in P

Solution: Take a language L which is not in $E = \cup_{c=1}^{\inf} TIME(2^{cn})$. Now consider the language $L' = \{1^m \mid m \in L\}$. Then L' is clearly in P/Poly, but its not in P. If it were decidable in $O(m^k)$, then we would decide L in time $O((2^n)^k)$, and so L would be in E. Our decision procedure works as follows:
- on input m of length $n = \log m$, we run the algorithm for L' on the input 1^m. This runs in time $O(m^k) = O((2^n)^k)$.

It remains to ensure that L' is decidable. To that end, all we need to do is to choose some $L \notin E$ which is decidable, for that makes L' trivially decidable: given an input, if it's not of the form 1^m, reject; otherwise, answer according to whether $m \in L$.

The existance of a decidable language $L \notin E$ is guaranteed by the time hierarchy theorem.

3. Describe an undecidable language that is in P/Poly but not in P

Solution: Consider the undecidable language HALT
HALT = $\{\langle M, w\rangle \mid M$ is a TM that halts on input $w\}$. Halt is undecidable.
Define $L' = \{1^m \mid m$ is the encoding of $\langle M, w\rangle \in HALT\}$. This is a unary encoding of HALT.
L' is still undecidable because HALT is undecidable, and it is just a unary encoding of HALT.
A language is in P/poly if it can be decided in polytime with a polynomial sized device. For L' we can provide the advice string for all unary strings upto length n, encoding whether each unary string 1^m corresponds to an instance $\langle M, w\rangle \in HALT$. This advice string is polynomial in size (it only needs O(n) bits for unary strings of length n).

Now L' is in P/poly since for each input length, we can have a polynomial sized advice string, which can be used to guide the machine to halt. Also, since it is polynomial sized, the verification can be done in polynomial time.

L' is not in P since if it were decidable in P, it could solve the Halting Problem. This would contradict the undecidability of $HALT$. Thus, L' is in $P/Poly$ but not in P.

4. A language L is in the class DP if and only if there are two languages L1 in NP and L2 in coNP such that L = L1 intersection L2.
SAT-UNSAT: Given two Boolean expressions phi and phi' (both in 3CNF). Is it true that phi is satisfiable and phi' is not?
Show that SAT-UNSAT is DP-Complete.

Solution: To show that SAT-UNSAT is DP-Complete, we need to show that it is in DP and that every language in DP is polynomial time reducible to SAT-UNSAT.

1. SAT-UNSAT is in DP:

L1 $\in$ NP and L2 $\in$ coNP such that L1 = $\{(\phi, \phi') \mid \phi$ is satisfiable$\}$ $\in$ NP and L2 = $\{(\phi, \phi') \mid \phi'$ is unsatisfiable$\}$ $\in$ coNP. Then it becomes trivial since SAT-UNSAT is exactly the intersection of L1 and L2. Since L1 and L2 are in NP and coNP respectively, their intersection is in DP.

2. SAT-UNSAT is DP-Hard:

To prove that SAT-UNSAT is DP-Hard, we reduce any language $L \in$ DP to SAT-UNSAT. Let $L = L1 \cap L2$, where $L1 \in$ NP and $L2 \in$ coNP. Then for any input $x \in L$, we construct two Boolean Formulae ϕ and ϕ' such that:

- $x \in L1 \iff \phi$ is satisfiable
- $x \in L2 \iff \phi'$ is unsatisfiable

This reduction ensures that $x \in L \iff (\phi, \phi') \in$ SAT-UNSAT.

Since such a reduction is possible in polynomial time for any $L \in$ DP, SAT-UNSAT is DP-Hard.
Therefore, SAT-UNSAT is DP-Complete.

5. The language CKT-SAT consists of all (strings representing) circuits that produce a single bit of output and that have a satisfying assignment. That is, a string representing an n-input circuit, C, is in CKT-SAT iff there exists u in $\{0,1\}^n$ such that C(u) = 1.

   Show that CKT-SAT is NP-Complete.

   Solution: To show that CKT is NP-Complete, we must show that it is in NP and that every language in NP is polynomial time reducible to CKT-SAT.

   1. CKT-SAT is in NP:

   We can easily build a Turing machine that can verify a satisfying assignment for a given circuit. Given a circuit C and an assignment u, the machine can evaluate C(u) and check if it is 1. To verify, the verifier is given a candidate assignment $u \in \{0,1\}^n$, and it compute C(u) by evaluating the circuit on u. If C(u) = 1, the verifier accepts, otherwise it rejects.

   Evaluating a circuit $C(u)$ takes time proportional to the size of C, which is polynomial in the size of the input. Therefore, CKT-SAT is in NP.

   2. CKT-SAT is NP-Hard:

   To prove that CKT-SAT is NP-Hard, we reduce SAT to CKT-SAT. In SAT, we are given a boolean formula $\phi(x_1, x_2, x_3, ..., x_n)$ in CNF, and we must decide if there exists a satisfying assignment $u \in \{0,1\}^n$ such that $\phi(u) = 1$. To transform SAT into CKT-SAT, construct a boolean circuit C that computes $\phi(x_1, x_2, ..., x_n)$:

   - represent ϕ as a circuit by encoding the CNF formula with logic gates (AND gates for clauses and OR gates for literals, and NOT gates for negations)
   - The transformation can be done in polynomial time because converting a CNF formula into a circuit representation is a polynomial time operation.

   If ϕ is satisfiable, then there exists an assignment u such that $\phi(u) = 1$, in which case the circuit C will output 1, and the instance of CKT-SAT corresponds to a YES instance.

   If ϕ is unsatisfiable, then there is no assignment u such that $\phi(u) = 1$, in which case the circuit C will output 0, and the instance of CKT-SAT corresponds to a NO instance.

   The reduction clearly happens in polynomial time since constructing C from ϕ involves creating a circuit of size polynomial in the size of ϕ. Therefore, CKT-SAT is NP-Hard.

6. Max-Sat

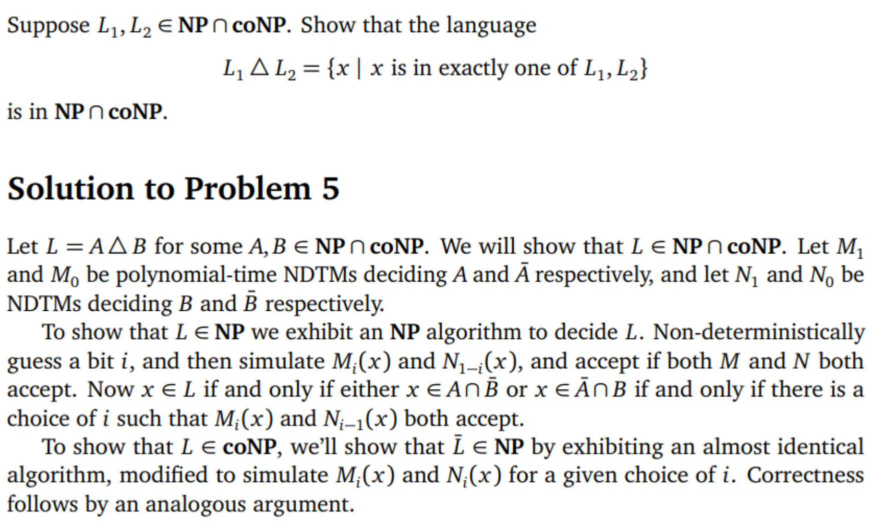
. In the MAX SAT problem we are given a formula φ in conjunctive normal form and we want to find the assignment of values to the variables that maximizes the number of satisfied clauses. (For example, if φ is satisfiable, the optimal solution satisfies all the clauses and the MAX SAT problem reduces to finding a satisfying assignment.) Consider the following decision problem: given a formula φ in conjunctive normal form and an integer k, determine if k is the number of clauses of φ satisfied by an optimal assignment.

- Prove that this problem is in **NP** if and only if **NP** $=$ *co***NP**.

  Solution Sketch. Suppose that MAX SAT is in **NP**, and let $V(\cdot,\cdot)$ be the verifier for MAX SAT. Then we can deduce that the co**NP**-complete problem UNSAT (where, given a CNF formula φ we want to decide if it is unsatisfiable) is also in **NP**: a witness that φ is unsatisfiable is a pair (k, y), where k is an integer smaller than the number of clauses of φ and y is such that $V((\varphi, k), y)$ accepts. But if a co**NP**-complete problem belongs to **NP**, it follows that *co***NP** $=$ **NP**.

  Suppose now that **NP** $=$ *co***NP**, and consider the language L that contains pairs (φ, k) such that at least k clauses of φ can be satisfied, and the language L' that contains the pairs (φ, k) such that it is impossible to satisfy k or more clauses of φ. By definition, L is in **NP**, and let $V()$ be its verifier; also L' is in co**NP**, and, by the assumption, also in **NP**, and let $V'()$ be its verifier. We can now put MAX SAT in **NP** by noting that a witness for (φ, k) $\in$MAX SAT is a pair (y, y') such that $V((\varphi, k), y)$ and $V'((\varphi, k + 1), y')$ both accept.

7. Another



Suppose $L_1, L_2 \in$ **NP** $\cap$ **coNP**. Show that the language

$$L_1 \triangle L_2 = \{x \mid x \text{ is in exactly one of } L_1, L_2\}$$

is in **NP** $\cap$ **coNP**.

Solution to Problem 5

Let $L = A \triangle B$ for some $A, B \in$ **NP** $\cap$ **coNP**. We will show that $L \in$ **NP** $\cap$ **coNP**. Let M_1 and M_0 be polynomial-time NDTMs deciding A and $\bar{A}$ respectively, and let N_1 and N_0 be NDTMs deciding B and $\bar{B}$ respectively.

To show that $L \in$ **NP** we exhibit an **NP** algorithm to decide L. Non-deterministically guess a bit i, and then simulate $M_i(x)$ and $N_{1-i}(x)$, and accept if both M and N both accept. Now $x \in L$ if and only if either $x \in A \cap \bar{B}$ or $x \in \bar{A} \cap B$ if and only if there is a choice of i such that $M_i(x)$ and $N_{i-1}(x)$ both accept.

To show that $L \in$ **coNP**, we'll show that $\bar{L} \in$ **NP** by exhibiting an almost identical algorithm, modified to simulate $M_i(x)$ and $N_i(x)$ for a given choice of i. Correctness follows by an analogous argument.

Solution:

8. Show that if P $=$ NP, then there is a language in EXP that requires circuits of size $2^n/n$

Solution: This statement essentially says the existence of an EXP-Hard language that cannot be efficiently represented as circuits.

The time hierarchy theorem tells us that there are languages in EXP that cannot be computed in subexponential time. Specifically, there exists a language $L \in DTIME(2^{O(n)})$ that cannot be computed in $DTIME(2^{o(n)})$.

By the connection between the time hierarchy theorem and circuit complexity, if a language an be decided in time $2^{o(n)}$, then it can be computed by circuits of size $2^{O(n)}$. Conversely, if a language requires $2^{\Omega(n)}$ time to decide, it will require circuits of size $2^{\Omega(n)}$ to compute.

By the Karp-Lipton theorem, if NP $\subseteq$ P/poly, then the polynomial hierarchy (PH) collapses to Σ_2^P.

We construct a specific language $L \in EXP$ that cannot be computed by small circuits. This language "diagonalizes" against all small circuits of size $\frac{2^n}{n}$

Assume P = NP, then NP $\subseteq$ P/poly. Then all languages in NP have polynomial sized circuits. By the Karp-Lipton theorem, PH collapses to P.

Assume for contradiction that L has circuits of size $\frac{2^n}{n}$. Then we could siulate L in $DTIME(2^{n/2})$, because we can use the circuit to decide L in smaller time by hardcoding the circuit's output. This contradicts the time hierarchy theorem which guarantees that $L \notin DTIME(2^{n/2})$.

Thus, L requires circuits of size $2^n/n$.

9. A language $L \subseteq \{0,1\}^*$ is sparse if there is a polynomial p such that $| L \cap \{0,1\}^n | \leq p(n)$ for every $n \in N$. Show that every sparse language is in P/poly

Solution: Let L be a sparse language. Then for each n, define the advice string a_n as follows:

- enumerate all strings in $L \cup \{0,1\}^n$ i.e., the set of all strings of length n that belong to L. Since L is sparse, the size of this set is at most p(n), where p(n) is a polynomial
- Let a_n be the concatenation of all strings in $L \cup \{0,1\}^n$, encoded in such a way that individual strings are distinguishable, (such as with a whitespace or prefix each string with its length in binary)

Then the total length of a_n is at most $p(n) \cdot n + p(n) \cdot \log p(n)$. This is polynomial in n, so the advice string is polynomial in size.

Then we construct a deterministic poly time turing machine M that decides membership in L using a_n as advice:

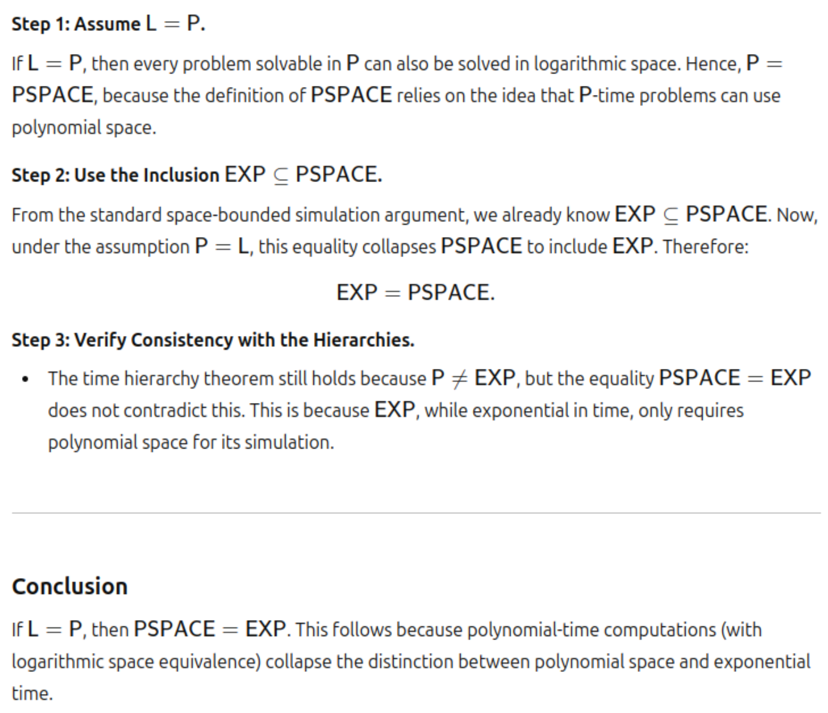
- On input x of length n, M retrieves the advice string a_n
- M parses a_n to reconstruct the set $L \cup \{0,1\}^n$ (using the defined encoding)
- M checks whether x is in this set. Since the cardinality is less than p(n), this takes polytime.

If $x \in L$, then M will accept, and if $x \notin L$, then M will reject. Since M runs in polynomial time, as checking membership and parsing the advice string are both polynomial time operations, L is in P/poly. Thus, every sparse language is in P/poly.

{ Just something - If a sparse language is NP-Complete, then P = NP:
We use the Mahaney's Theorem for this problem. It states that if any sparse language is NP-Complete, then P = NP. Also, if any sparse language is NP-Complete with respect to Turing reductions, then the polynomial-time hierarchy collapses to the $\triangle_2^P$. Since $P = NP$, and $P \subseteq P/poly$, then NP $\subseteq P/poly$. }

10. Show that if L = P, then PSPACE = EXP

Solution:



Step 1: Assume L = P.

If L = P, then every problem solvable in P can also be solved in logarithmic space. Hence, P = PSPACE, because the definition of PSPACE relies on the idea that P-time problems can use polynomial space.

Step 2: Use the Inclusion EXP ⊆ PSPACE.

From the standard space-bounded simulation argument, we already know EXP ⊆ PSPACE. Now, under the assumption P = L, this equality collapses PSPACE to include EXP. Therefore:

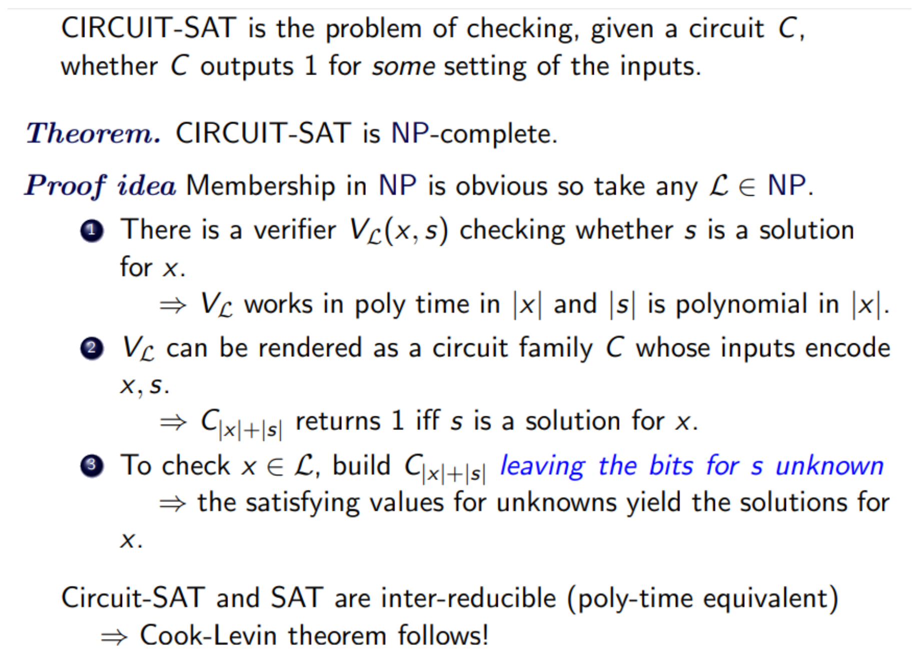
$$\text{EXP} = \text{PSPACE}.$$

Step 3: Verify Consistency with the Hierarchies.

- The time hierarchy theorem still holds because P ≠ EXP, but the equality PSPACE = EXP does not contradict this. This is because EXP, while exponential in time, only requires polynomial space for its simulation.

Conclusion

If L = P, then PSPACE = EXP. This follows because polynomial-time computations (with logarithmic space equivalence) collapse the distinction between polynomial space and exponential time.



CIRCUIT-SAT is the problem of checking, given a circuit C, whether C outputs 1 for *some* setting of the inputs.

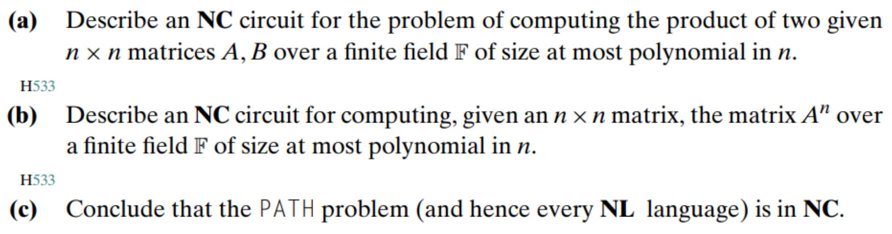
Theorem. CIRCUIT-SAT is NP-complete.

Proof idea Membership in NP is obvious so take any $\mathcal{L} \in$ NP.

1. There is a verifier $V_{\mathcal{L}}(x, s)$ checking whether s is a solution for x.
   $\Rightarrow$ $V_{\mathcal{L}}$ works in poly time in $|x|$ and $|s|$ is polynomial in $|x|$.
2. $V_{\mathcal{L}}$ can be rendered as a circuit family C whose inputs encode x, s.
   $\Rightarrow$ $C_{|x|+|s|}$ returns 1 iff s is a solution for x.
3. To check $x \in \mathcal{L}$, build $C_{|x|+|s|}$ *leaving the bits for s unknown*
   $\Rightarrow$ the satisfying values for unknowns yield the solutions for x.

Circuit-SAT and SAT are inter-reducible (poly-time equivalent)
$\Rightarrow$ Cook-Levin theorem follows!

11. Question:



(a) Describe an **NC** circuit for the problem of computing the product of two given $n \times n$ matrices A, B over a finite field $\mathbb{F}$ of size at most polynomial in n.

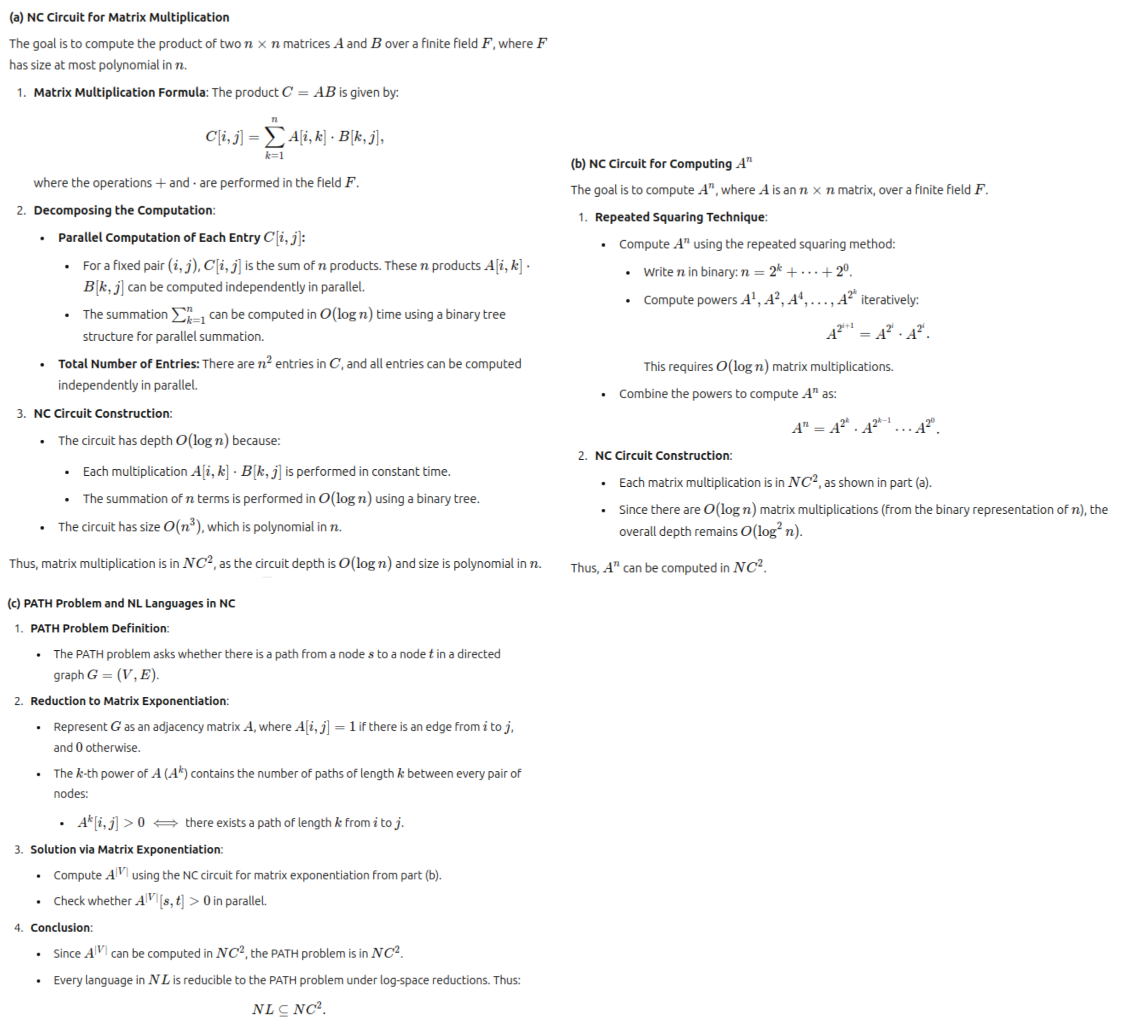
H533

(b) Describe an **NC** circuit for computing, given an $n \times n$ matrix, the matrix A^n over a finite field $\mathbb{F}$ of size at most polynomial in n.

H533

(c) Conclude that the PATH problem (and hence every **NL** language) is in **NC**.

Solution:



(a) NC Circuit for Matrix Multiplication

The goal is to compute the product of two $n \times n$ matrices A and B over a finite field F, where F has size at most polynomial in n.

1. **Matrix Multiplication Formula**: The product $C = AB$ is given by:

$$C[i, j] = \sum_{k=1}^{n} A[i, k] \cdot B[k, j],$$

   where the operations $+$ and $\cdot$ are performed in the field F.

2. **Decomposing the Computation**:
   - **Parallel Computation of Each Entry $C[i, j]$:**
     - For a fixed pair (i, j), $C[i, j]$ is the sum of n products. These n products $A[i, k] \cdot B[k, j]$ can be computed independently in parallel.
     - The summation $\sum_{k=1}^{n}$ can be computed in $O(\log n)$ time using a binary tree structure for parallel summation.
   - **Total Number of Entries:** There are n^2 entries in C, and all entries can be computed independently in parallel.

3. **NC Circuit Construction**:
   - The circuit has depth $O(\log n)$ because:
     - Each multiplication $A[i, k] \cdot B[k, j]$ is performed in constant time.
     - The summation of n terms is performed in $O(\log n)$ using a binary tree.
   - The circuit has size $O(n^3)$, which is polynomial in n.

Thus, matrix multiplication is in NC^2, as the circuit depth is $O(\log n)$ and size is polynomial in n.

(b) NC Circuit for Computing A^n

The goal is to compute A^n, where A is an $n \times n$ matrix, over a finite field F.

1. **Repeated Squaring Technique**:
   - Compute A^n using the repeated squaring method:
     - Write n in binary: $n = 2^k + \cdots + 2^0$.
     - Compute powers $A^1, A^2, A^4, \ldots, A^{2^k}$ iteratively:

$$A^{2^{i+1}} = A^{2^i} \cdot A^{2^i}.$$

       This requires $O(\log n)$ matrix multiplications.
   - Combine the powers to compute A^n as:

$$A^n = A^{2^k} \cdot A^{2^{k-1}} \cdots A^{2^0}.$$

2. **NC Circuit Construction**:
   - Each matrix multiplication is in NC^2, as shown in part (a).
   - Since there are $O(\log n)$ matrix multiplications (from the binary representation of n), the overall depth remains $O(\log^2 n)$.

Thus, A^n can be computed in NC^2.

(c) PATH Problem and NL Languages in NC

1. **PATH Problem Definition**:
   - The PATH problem asks whether there is a path from a node s to a node t in a directed graph $G = (V, E)$.
2. **Reduction to Matrix Exponentiation**:
   - Represent G as an adjacency matrix A, where $A[i, j] = 1$ if there is an edge from i to j, and 0 otherwise.
   - The k-th power of A (A^k) contains the number of paths of length k between every pair of nodes:
     - $A^k[i, j] > 0 \iff$ there exists a path of length k from i to j.
3. **Solution via Matrix Exponentiation**:
   - Compute $A^{|V|}$ using the NC circuit for matrix exponentiation from part (b).
   - Check whether $A^{|V|}[s, t] > 0$ in parallel.
4. **Conclusion**:
   - Since $A^{|V|}$ can be computed in NC^2, the PATH problem is in NC^2.
   - Every language in NL is reducible to the PATH problem under log-space reductions. Thus:

$$NL \subseteq NC^2.$$

12. Closure properties of P/poly

Solution: 1. Union:

If $L_1 \in$ P/poly and $L_2 \in$ P/poly, then there are polynomial time circuit families $C_n^{(1)}$ and $C_n^{(2)}$ that decide L_1 and L_2 respectively. We can construct a circuit C_n that decides $L_1 \cup L_2$ as follows:

1. Input x to C_n
2. Feed x to both $C_n^{(1)}$ and $C_n^{(2)}$
3. Output 1 if either $C_n^{(1)}$ or $C_n^{(2)}$ outputs 1, or more formally, $C_n(x) = C_n^{(1)}(x) \vee C_n^{(2)}(x)$

The size of C_n is the sum of the sizes of $C_n^{(1)}$ and $C_n^{(2)}$, which is polynomial in n. Therefore, $L_1 \cup L_2 \in$ P/poly.

2. Intersection

If $L_1 \in$ P/poly and $L_2 \in$ P/poly, then there are polynomial time circuit families $C_n^{(1)}$ and $C_n^{(2)}$ that decide L_1 and L_2 respectively. We can construct a circuit C_n that decides $L_1 \cap L_2$ as follows:

1. Input x to C_n
2. Feed x to both $C_n^{(1)}$ and $C_n^{(2)}$

3. Output 1 if both $C_n^{(1)}$ and $C_n^{(2)}$ output 1, or more formally, $C_n(x) = C_n^{(1)}(x) \wedge C_n^{(2)}(x)$

The size of C_n is the sum of the sizes of $C_n^{(1)}$ and $C_n^{(2)}$, which is polynomial in n. Therefore, $L_1 \cap L_2 \in \mathrm{P/poly}$.

3. Complement

If $L \in \mathrm{P/poly}$, then there is a polynomial time circuit family C_n that decides L. We can construct a circuit C'_n that decides $\overline{L}$ as follows:

1. Input x to C_n
2. Output 1 if C_n outputs 0, and output 0 if C_n outputs 1, or more formally, $C'_n(x) = \neg C_n(x)$

The size of C'_n is the same as the size of C_n, which is polynomial in n. Therefore, $\overline{L} \in \mathrm{P/poly}$.

4. Polytime Reducability

If $L_1 \in \mathrm{P/poly}$, and $L_1 \leq_p L_2$, i.e, L_1 is polytime reducible to L_2, then there is a polynomial time circuit family $C_n^{(1)}$ that decides L_1, and a polynomial time reduction f that maps instances of L_1 to instances of L_2. In other words, there exists a polytime computable function f such that $x \in L_1 \iff f(x) \in L_2$. Let $C_n^{(2)}$ be the circuit family that decides L_2. We construct a new circuit $C_n^{(1)}$ for L_1 as follows:

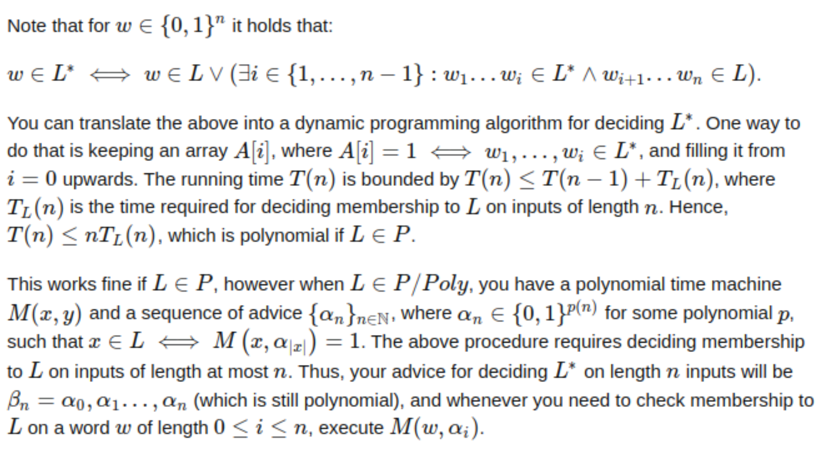
1. Input x to $C_n^{(1)}$
2. Compute $f(x)$ - this step takes polynomial time since f is polytime computable
3. Feed $f(x)$ to $C_n^{(2)}$ and output whatever $C_n^{(2)}$ outputs, or more formally, $C_n^{(1)}(x) = C_n^{(2)}(f(x))$

The size of $C_n^{(1)}$ is polynomial in n since it is constructed from $C_n^{(2)}$ and f, which are polynomial in size and polytime computable respectively. Therefore, $L_1 \in \mathrm{P/poly}$.

5. Kleene Star

We need to show that if $L \in \mathrm{P/poly}$, then $L^* \in \mathrm{P/poly}$. The class P/poly consists of languages decided by polynomial-sized circuit families. However, deciding L^* for arbitrary strings w involves checking whether w can be decomposed into substrings from L, which is computationally harder than the original membership problem for L.

While L^* can be computed for some languages in P/poly, it is not true in general. For example, consider the language $L = \{0,1\}^*$, which is in P/poly. The language L^* is the set of all strings over $\{0,1\}$, which is not sparse and cannot be computed by polynomial-sized circuits. In general, the construction of L^* depends on the structure of L, thus there is no guarantee that $L^* \in \mathrm{P/poly}$ for arbitrary $L \in \mathrm{P/poly}$.



Note that for $w \in \{0,1\}^n$ it holds that:

$$w \in L^* \iff w \in L \vee (\exists i \in \{1,\dots,n-1\} : w_1 \dots w_i \in L^* \wedge w_{i+1} \dots w_n \in L).$$

You can translate the above into a dynamic programming algorithm for deciding L^*. One way to do that is keeping an array $A[i]$, where $A[i] = 1 \iff w_1,\dots,w_i \in L^*$, and filling it from $i = 0$ upwards. The running time $T(n)$ is bounded by $T(n) \leq T(n-1) + T_L(n)$, where $T_L(n)$ is the time required for deciding membership to L on inputs of length n. Hence, $T(n) \leq nT_L(n)$, which is polynomial if $L \in P$.

This works fine if $L \in P$, however when $L \in P/Poly$, you have a polynomial time machine $M(x,y)$ and a sequence of advice $\{\alpha_n\}_{n \in \mathbb{N}}$, where $\alpha_n \in \{0,1\}^{p(n)}$ for some polynomial p, such that $x \in L \iff M\left(x, \alpha_{|x|}\right) = 1$. The above procedure requires deciding membership to L on inputs of length at most n. Thus, your advice for deciding L^* on length n inputs will be $\beta_n = \alpha_0, \alpha_1 \dots, \alpha_n$ (which is still polynomial), and whenever you need to check membership to L on a word w of length $0 \leq i \leq n$, execute $M(w, \alpha_i)$.

6. Concatenation

If $L_1 \in \mathrm{P/poly}$ and $L_2 \in \mathrm{P/poly}$, then there are polynomial time circuit families $C_n^{(1)}$ and $C_n^{(2)}$ that decide L_1 and L_2 respectively. For an arbitrary input x, to decide whether $x \in L_1 \circ L_2$, we would need to split x into two parts x_1 and x_2 such that $x = x_1 \circ x_2$, and then check if $x_1 \in L_1$ and $x_2 \in L_2$. However, this involves checking all possible splits of x, which could lead to an exponential blow-up in the circuit complexity.

Therefore, the concatenation of two languages in P/poly is not guaranteed to be in P/poly.