Question 1

For each of the following claims about the 'inner product mod 2" function $IP\colon \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$

$$IP(x,y) = x_1y_1 + x_2y_2 + \cdots + x_ny_n \bmod 2.$$

say if it is true or false (for all sufficiently large n). Prove your assertion (referring to results from the lecture notes if needed).

(a) IP has a width-4 read-once branching program when the input is read in order $x_1, y_1, \ldots, x_n, y_n$.

Solution: True. The state of the branching program at time $2t$ is determined by a single register z storing the value $x_1y_1 + \cdots + x_ty_t \bmod 2$. At time $2t+1$ another register x also stores the last seen x-value x_{t+1}. Specifically, the transitions $f_1, f_2, \ldots, f_{2n}$ are

$$f_1(\text{start}, x_1) = (0, x_1), \quad f_2((z,x), y_1) = (z + xy_1, 0), \quad f_3((z,x), x_2) = (z, x_2), \quad f_4((z,x), y_2) = (z + xy_2, 0)$$

and so on. At time $2n$ the register z contains the value $IP(x,y)$ and the program accepts iff this value is 1.

(b) IP has a width-$(2^n - 1)$ read-once branching program when the input is read in order $x_1, \ldots, x_n, y_1, \ldots y_n$.

Solution: False. If it did there would have to be two distinct strings $x \neq x'$ that reach the same state after the first n bits are read. Then for every y the branching program must output the same value on input (x,y) and (x',y). Let i be a position in which the two differ, namely $x_i \neq x'_i$. Let y be the string that has a 1 in position i and 0 everywhere else. Then $IP(x,y) = x_i \neq x'_i = IP(x',y)$, so the values $IP(x,y)$ and $IP(x',y)$ are different. Therefore the branching program cannot compute IP.

(c) IP has deterministic query complexity $2n$.

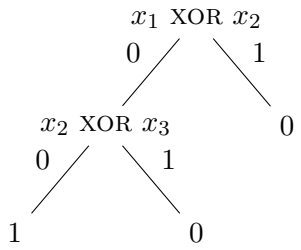
Solution: True. IP evaluated on the all-ones input has sensitivity $2n$ because flipping any variable changes the value of IP. As query complexity is lower bounded by sensitivity it must also be $2n$.

(d) IP has a depth-4 AND/OR circuit of size at most n^2.

Solution: False. If this was true for IP it would also be true for n-bit $PARITY$ because it is a restriction of IP, i.e., $PARITY(x) = IP(x, 11\cdots1)$. In Lecture 2 we showed that $PARITY$ requires depth-4 size $2^{\Omega(n^{1/3})}$, which is larger than n^2 asymptotically.

Question 2

A *parity decision tree (PDT)* is a generalized type of decision tree that can query parities of arbitrary subsets of the variables. For example this is a depth-2 PDT for the function "x_1, x_2, and x_3 are all equal".



Show that

(a) Every depth-d PDT can be computed by a $\mathbb{F}_2$-polynomial ($+$ is XOR, $\times$ is AND) of degree at most d.
(**Hint:** Modify the proof of Claim 4 from Lecture 4.)

Solution: First consider a "special" PDT that evaluates on 1 on exactly one path like in the example. This function is an AND of the XORs queried at each node, where each XOR is shifted by 1 if the answer is zero. For example, the function computed by the above PDT is $(x_1 + x_2 + 1)(x_2 + x_3 + 1)$. Viewed as a polynomial, this is a product of ℓ linear functions where ℓ is the length of the path, so it has degree ℓ. The function computed by a general PDT is a sum of such special decision trees, one for each path that leads to a 1-leaf. Therefore its degree cannot be larger than the length of the longest path, that is the PDT depth.

(b) AND of n inputs requires PDT depth n. (Use part (a).)

Solution: The $\mathbb{F}_2$-polynomial computing AND is $x_1 \cdots x_n$, so AND has degree n. Since this representation is unique any PDT computing AND must also have degree n. By part (a) it must have depth n.

(c) MAJORITY on n inputs requires PDT size $\Omega(2^{n^\epsilon})$ for some constant $\epsilon > 0$.
(**Hint:** Convert the PDT to an AND/OR/PARITY circuit.)

Solution: A size-s PDT can be simulated by a depth-3 AND/OR/PARITY circuit of size at most $s(n+1)$: Each path leading to a 1-leaf is an AND of at most n parities by the argument in part (a), and the PDT is an OR of at most s such paths. In Lecture 2 we showed that MAJORITY on n inputs requires depth-3 AND/OR/PARITY circuit size $2^{\Omega(n^{1/6})}$, so $s(n+1)$ must be at least as large as this number. Therefore $s \geq 1/(n+1) \cdot 2^{\Omega(n^{1/6})} = 2^{\Omega(n^{1/6})}$.

Question 3

Let $BPSAT$ be the decision problem whose input is a branching program B and whose YES instances are those B that accept at least one of their inputs. Let $ROBPSAT$ be the analogous decision problem for *read-once* branching programs. Argue that

(a) $BPSAT$ and $ROBPSAT$ are in NP.

Solution: The NP-relation consists of pairs (B, x) where B is a branching program (read-once in the case of $ROBPSAT$) and x is an input accepted by B. Checking whether B accepts x or not amounts to simulating B on x which can be done in polynomial time, in fact in linear time in your favorite programming language: The simulation keeps track of the state the branching program is in from left to right and accepts iff an accept state is reached.

(b) $ROBPSAT$ is in P.

Solution: A branching program accepts some input if and only if there exists a sequence of possible transitions from the start state to an accept state. Viewing the branching program as a directed graph with the edges representing possible transitions, checking whether B is a YES instance of $ROBPSAT$ amounts to verifying the existence of a path from the start state to the accept state in this graph. This can be done in linear time in any reasonable programming language via breadth-first or depth-first search (so in polynomial time on a Turing Machine).

(c) $BPSAT$ is NP-complete.
(**Hint:** Reduce from SAT to $BPSAT$.)

Solution: In Lecture 3 we showed that any DNF can be represented by a width-3 branching program. By the same reasoning this is true for CNF. As the CNF is read left to right, two of the states track whether the current clause has been satisfied. After the clause has been read, the branching program transitions to the remaining rejecting state and stays there if the clause evaluates to false. Unless the branching program terminates in this state the input is accepted.

This transformation from CNF to branching programs is efficient and gives a reduction from SAT to $BPSAT$: CNF instance ϕ is mapped to branching program B, and a candidate solution x for B is mapped back to

itself. In the notation of Lecture 6, $inst(\phi) = B$ and $sol(x) = x$. This reduction maps satisfying assignments of ϕ to satisfying assignments of B (as required by the definition) because ϕ and B compute the same function by construction.