1 Merlin in polynomial space

Question 1 By an analysis of the Shen-Shamir protocol, show that QBF can be decided by an IP protocol in which Merlin computes his answers in polynomial space (in the length of the input formula).

At each step, (the honest) Merlin has to produce a polynomial \( P(X) \) that is equal to \( Q_1X_1 \cdot Q_2X_2 \cdot \ldots \cdot Q_kX_k \cdot G \), where \( Q_1, Q_2, \ldots, Q_k \) are (pseudo-)quantifiers among \( \{\forall, \exists, R\} \) and \( G \) is a polynomial of polynomial degree. \( P(X) \) is given through its list of coefficients.

Instead, one can compute \( P(v) \) for \( d+1 \) distinct values of \( v \), and Lagrange interpolation to obtain the final coefficients. One can also let Merlin enumerate all possible answers, and simulate the rest of the protocol in order to give the best possible answer, without actually trying to be honest.

Question 2 Deduce that, if \( \text{PSPACE} \subseteq \text{P/poly} \), then \( \text{PSPACE} = \text{MA} \).

First, \( \text{MA} \subseteq \text{PSPACE} \) because \( \text{MA} \subseteq \text{AM} \subseteq \Pi^p_2 \subseteq \text{PH} \subseteq \text{PSPACE} \) (Babai’s Theorem 3.12, them Theorem 3.19 [second set of lecture notes], then definition of the polynomial hierarchy 4.1 and Proposition 4.6 [first set of lecture notes]).

Conversely, in order to show that \( \text{PSPACE} \subseteq \text{MA} \), it suffices to show that QBF is in MA. This is because QBF is \( \text{PSPACE} \)-complete (Stockmeyer-Meyer Theorem 2.21 [first set of lecture notes]) and MA is closed under polynomial time reductions.

In order to do so, we use the previous question, but we replace Merlin in the Shamir-Shen protocol by polynomial-sized circuits: Merlin only needs polynomial space, hence his job can be simulated by polynomial-sized circuits by the assumption \( \text{PSPACE} \subseteq \text{P/poly} \). In the MA protocol to be constructed, we will then ask Merlin to produce those circuits; then Arthur will simulate the Shamir-Shen protocol by himself, using those circuits instead of asking Merlin again.
The main subtlety in the first part of the question is that \( \text{PSPACE} \subseteq \text{P} \) means that any bit that is computable in polynomial space is computable by a polynomial-sized circuit. But Merlin has to produce a \( p(n) \)-bit string representing a polynomial \( P_i(X) \) at round \( i \) of the protocol (where \( p(n) \) is some polynomial in the input size, and \( i \) ranges from 0 to some polynomial \( p'(n) \)). We can encode this by \( p(n) \times p'(n) \) circuits \( C_{ij} \), \( i \) being the round number and \( j \) being the bit number in the \( p(n) \)-bit string representing \( P_i(X) \).

We solve QBF in \( \text{MA} \) as follows. Merlin’s task is to give the list of all circuits \( C_{ij} \). This has polynomial size by assumption. If the QBF instance is true, then Merlin can produce them. Then Arthur will simulate the Shamir-Shen protocol, replacing answers given in round \( i \) by Merlin, by strings obtained by concatenating the bits obtained by evaluating \( C_{i0}, C_{i1}, \ldots, C_{ip(n)} \) on the input formula and the current list of values for the variables whose values were given in previous rounds of the protocol. This only takes polynomial time, and Arthur must accept with probability \( 1 \).

If the QBF instance is false, no strategy for Merlin in the Shamir-Shen protocol can make Arthur accept with probability more than \( \frac{1}{2^{n\ell}} \) (where \( \ell \) is the chosen exponent on error for the Shamir-Shen protocol). So no choice of list of circuits by Merlin in round 1 of the \( \text{MA} \) protocol can make Arthur accept with probability more than \( \frac{1}{2^{n\ell}} \) again.

2 The Zachos-Heller theorem

Let \( \Sigma = \{0, 1\} \). All our random tapes \( r, r_1, r_2, \ldots \), are strings over \( \Sigma \).

Take \( L \in \text{BPP} \), so that:

— if \( x \in L \) then \( Pr_r[\mathcal{M}(x, r) \text{ accepts}] \geq 1 - 1/2^n \),

— if \( x \notin L \) then \( Pr_r[\mathcal{M}(x, r) \text{ accepts}] \leq 1/2^n \),

where \( \mathcal{M} \) is a deterministic Turing machine working in polynomial time \( p(n) \), and using \( q(n) \) random bits (meaning that the size of \( r \) is \( q(n) \)).

For a bit \( b \in \Sigma \), say that \( \mathcal{M}(x, r) = b \) to abbreviate « either \( b = 1 \) and \( \mathcal{M}(x, r) \) accepts, or \( b = 0 \) and \( \mathcal{M}(x, r) \) rejects ». \( \mathcal{M}(x, r) \neq b \) is the negation of \( \mathcal{M}(x, r) = b \). Let \( R_{xb} \) be the set of those \( r \) such that \( \mathcal{M}(x, r) \neq b \).

Let also \( b = (x \in L) \) mean « either \( b = 1 \) and \( x \in L \), or \( b = 0 \) and \( x \notin L \) », and \( b \neq (x \in L) \) be its negation.

**Question 3** Let \( L' \) be the language of all tuples \((x, b, H)\) such that \( R_{xb} \) has a collision for \( H \), where \( b \in \Sigma \) and \( H = (h_1, \ldots, h_\ell) \) is a tuple of linear hash functions from \( \Sigma^{q(n)} \) to \( \Sigma^{m'} \), and where \( \ell \) and \( m' \) are polynomials in the size \( n \) of \( x \), \( \ell \geq m' \), to be determined later. Show that \( L' \) is in \( \text{NP} \).
We guess the collision $r$, the tapes $r_1, \ldots, r_\ell$ with which $r$ is in collision, and we check in polynomial time that $\mathcal{M}(x, r) \neq b$, $\mathcal{M}(x, r_1) \neq b$, \ldots, $\mathcal{M}(x, r_\ell) \neq b$ and that $r_j \neq r$ and $h_j(r_j) = h_j(r)$ for every $j$.

**Question 4** We define the following algorithm. On input $x$, we draw $b$ and $H$ (as described above) at random, uniformly and independently. Then we test whether $(x, b, H) \in L'$. If so, we return the special symbol fail, otherwise we return $b$.

(a) Show that if $b \neq (x \in L)$, then that algorithm must return fail… under a constraint on $n$, $\ell$, and $m'$ that you will give explicitly. We will name that constraint (A).

If $b \neq (x \in L)$, for example $x \in L$ but $b = 0$, so that $\mathcal{M}(x, r) \neq b$ with probability at least $1 - 1/2^n$, then $R_{xb}$ will have a collision for $H$ by Sipser II, as soon as:

\[(A) \quad (1 - 1/2^n)2^{q(n)} > \ell.2^{m'}.

If it has a collision for $H$, then $(x, b, H)$ is in $L'$, so we must return fail.

(b) Show that if $b = (x \in L)$, then the probability that the algorithm returns fail is smaller than or equal to $1/2^{\ell - m' + 1} \ldots$ under a constraint on $n$, $\ell$, and $m'$ that you will give explicitly. We will name that constraint (B).

If $b = (x \in L)$, then $\mathcal{M}(x, r) \neq b$ with probability at most $1/2^n$. By Sipser I, the probability that $R_{xb}$ has a collision for $H$ is at most $1/2^{\ell - m' + 1}$, provided $1/2^n.2^{q(n)} \leq 2^{m' - 1}$, namely:

\[(B) \quad q(n) - n \leq m' - 1.

With probability at least $1 - 1/2^{\ell - m' + 1}$, there will be no collision, so $(x, b, H)$ will not be in $L'$, hence the algorithm will return $b$, not fail.

(c) We simply take $\ell = m'$. Show that, for $n$ large enough, one can find $m'$ so that (A) and (B) are satisfied, and such that $m'$ is bounded by a polynomial in $n$.

To satisfy (B) we can just take $m' = q(n) - n + 1$, and that is polynomial. We check that (A) holds when $n$ is large enough: (A) is equivalent to $(1 - 1/2^n) > (q(n) - n + 1)2^{-n+1}$, whose left-hand side tends to 1, and whose right-hand side tends to 0.

**Question 5** Conclude that BPP is included in the class $\text{ZPP}^\text{NP}$ of languages that can be decided in expected polynomial time with zero error, on a randomized Turing machine with access to an NP oracle.

When $n$ is large enough, we run the algorithm of the previous question for as long as it returns fail. Once it returns some other value $b$, we stop and return $b$. By **Question 4** (a), since it did not return fail, we must have $b = (x \in L)$, so this algorithm is always correct.
The probability that the algorithm of Question 4 returns fail is at most $Pr_{b,H}[b \neq (x \neq L)] \times 1 + Pr_{b,H}[b = (x \neq L)] \times 1/2^{\ell - m' + 1} = 1/2(1 + 1/2^{\ell - m' + 1}) = 3/4$ (since $\ell = m'$). Hence the probability $p$ of terminating (not failing) at each turn of the loop is at least 1/4. Therefore our new algorithm, which repeats Question 4 until it does not fail, will stop in at most 4 turns of the loop on average.

When $n$ is not large enough, we simply use a precompiled table of answers, and return the tabulated answer for $x$.

3 \hspace{1cm} \text{EXPSPACE does not have polynomial circuits}

We fix $k \in \mathbb{N}$. For each $n \in \mathbb{N}$, let $\mathcal{C}_n$ be the set of all circuits with $n$ inputs of size exactly $n^{k+1}$ (yes, the same $n$). As usual, circuits are represented by their netlists, written in binary.

Question 6 Let us fix $n \in \mathbb{N}$. We enumerate the $n$-bit words as $x_1, \ldots, x_{2^n}$; $x_i$ is just $i \mod 2^n$ written in binary. Let $R^n_0 \overset{\text{def}}{=} \mathcal{C}_n$. By induction on $i \in \{1, \ldots, 2^n\}$, we say that:

- $x_i$ is in if and only if $C[x_i] = 0$ for at least half of the circuits $C$ in $R^n_{i-1}$;
- if so, then $R^n_i$ is the collection formed by the remaining circuits $C$ in $R^n_{i-1}$, namely those such that $C \in R^n_{i-1}$ and $C[x_i] = 1$;
- if, on the other hand, $x_i$ is out (namely, not in), then $R^n_i$ is the collection of all the circuits $C \in R^n_{i-1}$ such that $C[x_i] = 0$.

Let $L^n$ be the set of $n$-bit words that are in. Show that $R^n_i = \emptyset$ for every $i \geq n^{k+1} + 1$.

By induction on $i$, the cardinality of $R^n_i$ is at most $2^{n^{k+1}-i}$, since increasing $i$ by 1 divides at by at least two. Hence the cardinality of $R^n_{n^{k+1}+1}$ is at most one, and therefore $R^n_{n^{k+1}+1}$ is empty. (Its cardinality is strictly less than half of that of $R^n_{n^{k+1}+1}$.)

Question 7 Let $n$ be so large that $n^{k+1} + 1 \leq 2^n$. Let $C \in \mathcal{C}_n$. $C$ is right on $x_i$ if and only if $C[x_i] = 1$ and $x_i \in L^n$, or $C[x_i] = 0$ and $x_i \notin L^n$. Show that, for every $i \in \{0, \ldots, 2^n\}$, $R^n_i = \{C \in \mathcal{C}_n \mid C$ is right on $x_1, \ldots, x_i\}$. Conclude that there is a word $x$ of length $n$ on which $C$ is not right; so no circuit $C \in \mathcal{C}_n$ can decide $L^n$.

We prove the claim by induction on $i$. For $i = 0$, this boils down to the fact that $R^n_0 = \mathcal{C}_n$ is the collection of all circuits (namely, those that are right on every word from the empty set). The proof only depends on the definition of $R^n_i$, not on the first clause of the definition Question 6 stating whether $x_i$ is in or out.

Let $i \geq 1$. If $C \in R^n_i$, then by definition $C \in R^n_{i-1}$ (hence $C$ is right on $x_1, \ldots, x_{i-1}$ by induction hypothesis) and $C[x_i] = 1$ if $x_i$ is in (i.e., in $L^n$; second clause of the definition), $C[x_i] = 0$ if $x_i$ is out (namely, outside $L^n$; third clause), so in any case $C$ is right on $x_i$.

Conversely, if $C$ is right on $x_1, \ldots, x_{i-1}, x_i$, then by induction hypothesis $C$ is in $R^n_{i-1}$, and is right on $x_i$. If $x_i$ is in, namely in $L^n$, then since $C$ is right on $x_i$ we have $C[x_i] = 1$, so by the second clause $C$ is in $R^n_i$. If $x_i$ is out, namely outside $L^n$,
then since $C$ is right on $x_i$, we have $C[x_i] = 0$, so by the third clause, $C$ is also in $R^n_i$.

We conclude. By Question 6, $R^n_i$ is empty for, say, $i \overset{\text{def}}{=} n^{k+1} + 1$. (This is possible since $n^{k+1} + 1 \leq 2^n$, by assumption.) Hence the collection of circuits $C \in \mathcal{C}_n$ that are right on $x_1, \ldots, x_i$ is empty. Therefore, for every $C \in \mathcal{C}_n$, $C$ must fail to be right on at least one of the words $x_1, \ldots, x_i$.

**Question 8** Let $L \overset{\text{def}}{=} \bigcup_{n \in \mathbb{N}} L^n = n$. Show that $L \in \text{PSPACE}$.

We simply follow the definition of $L^n$, where $n$ is the length of the input $x$.

This is where we first require that our enumeration $x_i$ of the words of length $n$ is obtained by letting $x_i$ be simply $i \mod 2^n$ written in binary. This way, we can enumerate words by simply incrementing from 0. Also, the input $x$ will be equal to some $x_i$, where $i$ is simply $x$ interpreted as a number in binary (or $2^n$ if $x$ is all zeroes).

— If $i \geq n^{k+1} + 2$, then $R^n_{i-1}$ is empty by Question 6, so $x_i$ is in, and we must accept. Note that $n^{k+1}+2$ is computable in polynomial time, hence in polynomial space.

— Otherwise, we compute whether $x_i$ is in by enumerating the circuits $C$ in $R^n_{i-1}$, and counting how many of them are such that $C[x_i] = 0$, and how many there are in $R^n_{i-1}$. This requires two counters of size at most $n^{k+1}$. The challenge is to enumerate the circuits in $R^n_{i-1}$, without keeping them in memory from one iteration to the next.

For now, let us observe that the space needed to test whether $x_i$ is in is:

$$\text{Space}(x_i \text{ in } ?) = 2n^{k+1} + \text{Space}(R^n_{i-1}),$$

where $\text{Space}(R^n_{i-1})$ is the space needed to enumerate the circuits in $R^n_{i-1}$.

— In general, to enumerate the circuits in $R^n_i$ ($i \geq 1$), we simply enumerate all circuits $C$ in $\mathcal{C}_n$, using a counter of size $n^{k+1}$, and the question reduces to checking whether $C \in R^n_i$. Hence:

$$\text{Space}(R^n_i) = n^{k+1} + \text{Space}(C \in R^n_i?),$$

where $\text{Space}(C \in R^n_i?)$ is the space needed to test whether $C$ is in $R^n_i$.

— In order to test whether $C \in R^n_i$ ($i \geq 1$), we test recursively whether $C \in R^n_{i-1}$, and if so, we accept if $C[x_i] = 1$ and $x_i$ is in, or if $C[x_i] = 0$ and $x_i$ is out, and we reject otherwise. The computation of $C[x_i]$ takes polynomial time, hence polynomial space (precisely, $an^{k+1}$ for some constant $a$). Moreover, the space can be reused between recursive calls.

Hence:

$$\text{Space}(C \in R^n_i?) = an^{k+1} + b + \text{Space}(x_i \text{ in } ?),$$

for some constant $b$. 5
The three equations above yield the following recurrence equation:

\[ \text{Space}(x_i \text{ in } ?) = (a + 3)n^{k+1} + b + \text{Space}(x_{i-1} \text{ in } ?) \]

for every \( i \geq 2 \). When \( i = 1 \), \( R^n_{i-1} \) is \( \mathcal{C}_n \) by definition, so \( \text{Space}(R^0_n) = n^{k+1} \), and therefore \( \text{Space}(x_1 \text{ in } ?) = 3n^{k+1} \).

Summing up, we obtain that \( \text{Space}(x_i \text{ in } ?) = O(i \cdot n^{k+1}) \). Since \( i < n^{k+1} + 2 \) in the only important case, that is an \( O(n^{2k+2}) \).

Note that it is necessary to distinguish between the cases \( i \geq n^{k+1} + 1 \) and \( i < n^{k+1} + 1 \). If we did the recursive analysis leading to \( \text{Space}(x_i \text{ in } ?) = O(in^{k+1}) \) for all values of \( i \), we would obtain a procedure that works in space \( O(2n^{k+1} \cdot n^{k+1}) \), which is exponential, not polynomial.

Let \( \text{SIZE}(p(n)) \) denote the class of languages decided by families of circuits of size \( p(n) \), for any function \( p \). (They are defined analogously as for \( \text{P/poly} \).) \( L \) is in \( \text{PSPACE} \) by Question 8, but not in \( \text{SIZE}(n^{k+1}) \) by Question 7. Hence we obtain Kannan’s theorem:

**Theorem 1** For every \( k \in \mathbb{N} \), \( \text{PSPACE} \not\subseteq \text{SIZE}(n^{k+1}) \).

Exponential in \( n \) means \( O(2^{p(n)}) \) for some polynomial \( p \). \( \text{EXPSPACE} \) is the class of languages that one can decide on a deterministic Turing machine in exponential time.

**Question 9** Show that the complement \( \overline{L} \) of \( L \) is the padding of some language \( L' \), namely: the words of length \( n \) in \( \overline{L} \) are exactly the words \( x \) that we can write as a string of \( n - f(n) \) zero bits followed by an \( f(n) \)-bit word in \( L' \), for \( n \) large enough; \( f(n) \) is a function that you will make precise, and which is an \( O(\log n) \). We recall that \( x_i \) is the word obtained by writing \( i \mod 2^n \) in binary, and we agree that we write bits from left to right, so that the numbers less than or equal to, say, \( 2^n \), have \( n - m \) leading zero bits.

Clearly, \( L' \) will have to be the set of \( O(\log n) \)-sized suffixes of words of \( \overline{L} \), but for the claim to hold, we have to show that all the words of length \( n \) in \( \overline{L} \) have \( n - O(\log n) \) zero bits.

To show this, we recall that by Question 6, \( R^n_i \) is empty for every \( i \geq n^{k+1} + 1 \). By the first clause in the definition of \( L^{=n} \), \( x_i \) is then in \( L^{=n} \) for every \( i \geq n^{k+1} + 2 \) : indeed, at least half of the circuits \( C \) in \( R^n_{i-1} = \emptyset \) satisfy \( C[x_i] = 0 \) (or any other property for that matter).

It follows that no \( x_i \) of length \( n \) is in \( \overline{L} \) for any \( i \geq n^{k+1} + 2 \). In particular, every \( x_i \) of length \( n \) in \( \overline{L} \) can be written on just a suffix of \( \lceil \log_2(n^{k+1} + 2) \rceil \) bits, preceded by \( n - \lceil \log_2(n^{k+1} + 2) \rceil \) zero bits (for \( n \) large enough so that \( n^{k+1} + 2 \leq 2^n \)).

We have \( f(n) \overset{\text{def}}{=} \lceil \log_2(n^{k+1} + 2) \rceil \).

**Question 10** Show that \( L' \in \text{EXPSPACE} \).
This is almost Proposition 3.8 of the first part of the lecture notes (nl.pdf), modulo the replacement of \# by 0 (but Proposition 3.8 requires \# to be a fresh symbol).

Let \( g \) be a function such that \( f(g(n)) \geq n \) (and \( g(n) \geq n \)) for every \( n \). Since \( f(m) = O(\log m) \), \( g(n) \) is an exponential of \( n \).

In order to decide whether \( x \) (of length \( n \)) is in \( L' \), we pad \( x \) with \( g(n) - n \) leading zero bits, where \( g(n) \geq n \) is to be determined later, and we test whether the result \( 0^{g(n) - n}x \) is in \( \overline{L} \). This is correct if the \( f(g(n)) \)-length suffix of \( 0^{g(n) - n}x \) is long enough to contain all the bits of \( x \), namely if \( f(g(n)) \geq n \). It suffices to take \( g(n) \) equal to the least \( m \) such that \( f(m) \geq n \). Since \( f(m) = O(\log m) \), \( g(n) \) is an exponential of \( n \) (precisely, \( O(2^n/(k+1)) \)).

By Question 8, deciding whether \( 0^{g(n) - n}x \) is in \( \overline{L} \) takes polynomial space in \( g(n) \) (an exponential of \( n \)), to which we add the space needed to construct \( 0^{g(n) - n}x \) (\( g(n) \), an exponential).

**Question 11** Show that \( L' \not\in \text{P/poly} \).

Let us imagine that there are polynomial-sized circuits \( C_m \) deciding \( L' \), namely of size \( q(m) \), for some polynomial \( q \). Then we can decide whether \( x \in L \) by checking whether the \( n - f(n) \) first bits of \( x \) are 0, and whether the \( f(n) \) remaining bits are accepted by \( C_{f(m)} \). In other words, \( \overline{L} \) would have circuits (on inputs of length \( n \)) that:

- do a « nor » operation on the first \( n - f(n) \) bits, returning 1 if and only if they are 0;
- use \( C_{f(m)} \) on the remaining \( f(n) \) bits;
- do an « and » on the results of the previous two circuits.

This produces a circuit of size \( O(n - f(n)) + q(f(n)) + O(1) = O(n) \), since \( f(n) = O(\log n) \) (and therefore \( q(f(n)) \) is a \( O \) of some power of \( \log n \)). The circuit for \( L \) is obtained by adding one negation gate to the output of that circuit for \( \overline{L} \).

Whatever \( k \) we have chosen, the size of the circuit is an \( O(n^{k+1}) \), and we have shown in Question 7 that \( L \) cannot be decided by circuits of size \( n^{k+1} \), leading to a contradiction.

Hence we obtain the following theorem.

**Theorem 2** \( \text{EXPSPACE} \not\subseteq \text{P/poly} \).

4 \( \text{MA}_{\text{EXP}} \) does not have polynomial circuits

Let \( \text{MA}_{\text{EXP}} \) be defined just like \( \text{MA} \), except that Arthur works in exponential time and Merlin can give answers of exponential length. Explicitly, \( L \in \text{MA}_{\text{EXP}} \) if and only if there is a language \( D \in \text{P} \) such that for every input \( x \), of length \( n \):

- if \( x \in L \), then there is a \( y \) of length \( 2^{g(n)} \) such that \( Pr_r [x\#y\#r \in D] \geq 2/3 \);
— if $x \notin L$, then for every $y$ of length $2^{p(n)}$, $Pr_r [x \# y \# r \in D] \leq 1/3$;

where the probabilities are taken on all the random tapes $r$ of length $2^{q(n)}$, and $p(n)$ and $q(n)$ are polynomials. For simplicity, we only consider an error of $1/3$.

**Question 12** Show that $\text{PSPACE} \subseteq \text{MA}$ implies $\text{EXPSPACE} \subseteq \text{MA}_{\text{EXP}}$.

By padding (Proposition 3.8 in the first part of the lecture notes, \textit{nI. pcp}). Let $L \in \text{EXPSPACE}$, say decided in space $2^{p(n)}$, for some polynomial $p$. Then the language $L'$ of words $x \# 2^{p(n)} - n$ can be decided in linear space, and is in particular in $\text{PSPACE}$. By assumption, $L'$ is in $\text{MA}$, and it follows that, unfolding the definition, $L$ is in $\text{MA}_{\text{EXP}}$.

**Question 13** Using Theorem 2 and Question 2, among other things, deduce that $\text{MA}_{\text{EXP}} \not\subseteq P/\text{poly}$. (That is an improvement over Theorem 2.)

The main new thing to prove is that $\text{PSPACE} \subseteq \text{MA}_{\text{EXP}}$. This is easy: $\text{PSPACE} \subseteq \text{EXPTIME}$, and EXPTIME is included in $\text{MA}_{\text{EXP}}$, because any language decided in exponential time can be decided by an $\text{MA}_{\text{EXP}}$ protocol in which Arthur simply solves the question in exponential time, asks a dummy question, ignores Merlin’s answer, and returns the answer to the question.

Now, we have two cases:

— If $\text{PSPACE} = \text{MA}$, then by Question 12, $\text{EXPSPACE} \subseteq \text{MA}_{\text{EXP}}$. If we had $\text{MA}_{\text{EXP}} \subseteq P/\text{poly}$ it would follow $\text{EXPSPACE} \subseteq P/\text{poly}$, which would contradict Theorem 2.

— If $\text{PSPACE} \neq \text{MA}$, then by the contrapositive of Question 2, $\text{PSPACE} \not\subseteq P/\text{poly}$. We have argued above that $\text{PSPACE} \subseteq \overline{\text{MA}_{\text{EXP}}}$, so $\text{MA}_{\text{EXP}} \not\subseteq P/\text{poly}$.