Interactive Proofs, MA; RL and NL

All written documents allowed. No access to Internet, no mobile phones allowed.
Questions are graded according to their difficulty. I have given an indication of the number of lines my solution has for each question, between square brackets. But beware that difficulty and length may be uncorrelated. My solution to the whole exam is [102 lines] long.

A final note: it may (and will) happen that the key trick in solving one question was introduced in one of the previous questions.

1 Interactive Proofs and MA

[35 lines total for this part]

1. [2 lines] Let $p$ be a prime number, and $P(X)$ be a polynomial in one variable $X$ over $\mathbb{Z}/p\mathbb{Z}$, of degree $d \ll p$. Assume $P(X)$ is given implicitly: all we have is an oracle $O$ that computes $P(v)$ for any input $v \in \mathbb{Z}/p\mathbb{Z}$. Show that we can compute the coefficients $a_k$ of $X^k$ in $P(X)$, $0 \leq X \leq d$, in time polynomial in $d$ and $\log_2 p$ on a deterministic Turing machine with oracle $O$.

Direct application of Lagrange interpolation, using the fact that all arithmetic operations mod $p$ are in polynomial time.

If one does not remember the explicit formulae for the Lagrange interpolants, one can solve for the unknown coefficients using Gaussian elimination (it is a linear system) . . . but the existence and uniqueness of the solution is only guaranteed because of Lagrange interpolation, which could not be avoided here.

2. [10 lines] Reexamine the Shen protocol (or the Shamir protocol, if you prefer), and show that if Merlin can convince Arthur that the input QBF formula is true, then he can do so in in polynomial space. In other words, show that we can implement the Merlin function $M$ in the definition of the IP protocol for QBF by a polynomial-space computable function. (Note that the “obvious” way of producing the polynomial that Merlin submits to Arthur at each step is not polynomial space.)

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 $Q_1$, $Q_2$, . . . , $Q_k$ are (pseudo-)quantifiers among
\{\forall, \exists, R\} and \(G\) is a polynomial of polynomial degree. \(P(X)\) is given through its list of coefficients.

Computing \(P(X)\) by recursion on \(k\) does not lead to a polynomial space computation, since intermediate polynomials are degree \(\leq d\) polynomials (where \(d\) is polynomial in the input length \(n\)) in up to \(n\) variables: this means up to \((d + 1)^n\) coefficients, which is not polynomial in \(n\).

Instead, one can compute \(P(v)\) for \(d + 1\) distinct values of \(v\), and use Question 1 (Lagrange interpolation) to obtain the final coefficients.

3. [20 lines] If \(\text{PSPACE} \subseteq \text{P/poly}\), then the above shows that we can replace Merlin, in an interactive proof of QBF, by a polynomial-sized circuit. Make this precise, and show that, under this assumption, QBF is solvable in \(\text{MA}\).

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 \(\frac{1}{2^{\ell}}\) (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 \(\frac{1}{2^{\ell}}\) again.

4. [3 lines] Conclude that if \(\text{PSPACE} \subseteq \text{P/poly}\), then \(\text{PSPACE} = \text{MA}\).

We know that \(\text{MA} \subseteq \text{AM} \subseteq \Pi^p_2 \subseteq \text{PSPACE}\). Conversely, if \(\text{PSPACE} \subseteq \text{P/poly}\), then QBF is in \(\text{MA}\). But QBF is \(\text{PSPACE}\)-complete (Wrathall) and \(\text{MA}\) is closed under polynomial-time reductions.
2 RL and NL

[67 lines total for this part]

RL (randomized logspace) is the class of all languages \( L \) such that there is a randomized Turing machine \( M \), working in space \( O(\log n) \), that terminates with probability 1, and such that:

- If \( x \in L \), then \( Pr_r[M(x, r) \text{ accepts}] \geq \frac{1}{2} \);
- If \( x \not\in L \), then \( M(x, r) \) accepts for no \( r \) (it may reject, or fail to terminate—with zero probability).

As for ZPP, we do not require \( M \) to terminate on all inputs, just on a subset of measure 1. We also have the same measure-theoretic concerns as for ZPP, but will assume them solved.

Beware of a counter-intuitive fact: it will become apparent from the last questions below that we cannot require \( M \) to terminate in much less than exponential average time, while the analogy with NL would lead us to think that polynomial time would be enough. (The status of the corresponding class RLP of all problems solvable in logspace and polynomial time is not known.)

One can replace the \( \frac{1}{2} \) above by any constant \( \epsilon \) (\( 0 < \epsilon < 1 \)), or by any function of the form \( 1 - \frac{1}{2^{p(n)}} \), where \( p \) is a non-negative polynomial. As with RP, this is by repeating experiments. In the second case, the counter used takes only logarithmic space.

5. [9 lines] Show that \( RL \subseteq NL \).

Take an RL machine deciding \( L \), and decide to guess the bits instead of drawing them at random. By assumption, if \( x \in L \), then most of the branches will accept, in particular there is at least one. If \( x \not\in L \), then none accepts and most terminate, so most reject.

The resulting NL machine runs in space \( k \log n \) for some \( k \), but may fail to terminate. As in the lectures, we know that any run of more than \( a^{k \log n} = n^{k \log a} \) (where \( a \) is the alphabet size) will visit the same configuration twice. So we can stop any run when it exceeds \( n^{k \log a} \) steps, and reject. This requires an extra counter of size \( k \log n \).

6. [7 lines] Let \( G = (V, E) \) be a directed graph, \( s \) and \( t \) be two vertices of \( G \) (i.e., \( s, t \in V \)). Let \( N \) be the number \( \#V \) of vertices of \( G \). Show that the following repeated random walk algorithm \( A \), taking \( G \) as input, has the following properties:

- **P1** \( A \) runs in \( O(\log n) \) space, where \( n \) is the size of \( G \);
- **P2** If \( t \) is reachable from \( s \) in \( G \), then \( A \) accepts with probability 1;
- **P3** If \( t \) is not reachable from \( s \) in \( G \), then \( A \) does not terminate.
1 $v := s$;
2 for $i := 0$ to $N$ {
3     if $v = t$ then accept;
4     else if $v$ has no successor then break; // exit the loop (goto 10)
5     else {
6         pick a successor vertex $v'$ of $v$ in $G$ at random uniformly;
7         $v := v'$;
8     }
9 }
10 goto 1 // start again from $s$

For now, you shall assume that: (∗) there is a way of picking a successor vertex $v'$ of $v$ in $G$ at random uniformly in finite average time and using $O(\log n)$ space. We shall return to this in Question 7.

$P1$ and $P3$ are clear. For $P2$, if there is a path from $s$ to $t$, then necessarily there is one of length at most $N$. The probability that the random walk (lines 2–9) finds this precise path is $\frac{1}{N^N}$. This is small, but non-zero.

So the probability that the path is not found in $K$ outer loops (lines 1-10) is at most $(1 - \frac{1}{N^N})^K$. Since $\frac{1}{N^N}$ is non-zero, this tends to zero when $K \rightarrow +\infty$. So the probability that the path is not found in any number of loops is 0. This proves $P2$.

7. [5 lines] Why is assumption (∗) valid? I.e., propose a algorithmic way of picking $v'$ at random so that (∗) holds. If you don’t see the difficulty, train yourself on the following question first: how do you pick an element from a set of 3 possible successors, knowing that a randomized Turing machine only draws bits at random?

Number the bits on, say, $k$ bits, and do reject sampling: draw a $k$ bit string uniformly at random, until we find one that is one of the $N_v$ successors.

This terminates in an average number of draws $\sum_{i=0}^{+\infty} (i + 1)(1 - \frac{N_v}{2^k})^i \frac{N_v}{2^k} = \frac{2^k}{N_v}$ steps, and therefore in finite average time. This also clearly uses only logarithmic space.

8. [14 lines] Modify Algorithm $\mathcal{A}$ by adding a counter, so that the resulting algorithm $\mathcal{A}'$ satisfies:

$P1mod$ $\mathcal{A}'$ runs in space $O(\log n)$ plus the size of the counter;

$P1bis$ The space used by the counter is bounded by some polynomial $p(n)$ (yes, this means the counter is too big for our final purpose, but you should not be able to do better at this point);

$P2mod$ If $t$ is reachable from $s$ in $G$, then $\mathcal{A}$ accepts with probability at least $\frac{1}{2}$;
**P3mod** If \( t \) is not reachable from \( s \) in \( G \), then \( A \) rejects.

If \( t \) is reachable from \( s \) in \( G \), then the probability that \( A \) will find a given, fixed path from \( s \) to \( t \) in \( K \) turn of the outer loop is at least \( 1 - (1 - \frac{1}{N^K})^K \). We have seen that this tended to 1 when \( K \) tended to \(+\infty\).

Let us find \( K \) so that \( 1 - (1 - \frac{1}{N^K})^K \) is greater than or equal to \( \frac{3}{4} \). Equivalently, \( 1 - (1 - \frac{1}{N^K})^K \leq \frac{1}{2} \), i.e., \( K \log(1 - \frac{1}{N^K}) \leq -\log 2 \), i.e.,

\[
K \geq \frac{\log 2}{-\log(1 - \frac{1}{N^K})}
\]

The right-hand side (call it \( K_{\text{min}} \)) is an \( O(N^N) \).

\( A' \) works as \( A \), except it runs through the outer loop 1–10 at most \( K_{\text{min}} \) times. This will ensure \( P2\text{mod} \).

We reserve a counter for this. It requires space a logarithm of its maximum value \( O(\log N_N) = O(N \log N) \); this is polynomial. This establishes \( P1\text{bis} \).

When the counter reaches its maximum value, we let \( A' \) stop, and reject. This ensures that \( P3\text{mod} \) is satisfied.

Finally, \( P1\text{mod} \) is clear.

9. [6 lines] Imagine we wish to repeat some task \( T \) \( 2^{p(n)} \) times. One possibility is to maintain a \( p(n) \)-bit counter. If we have access to a source of randomness, we may instead repeat the following:

(a) Draw \( p(n) + 2 \) bits at random uniformly.
(b) If all of them are 0, then stop.
(c) Otherwise, do \( T \) and go back to (a)

Show that this will execute task \( T \) at least \( 2^{p(n)} \) times, with probability at least \( \frac{3}{4} \). (We assume \( p(n) \) tends to infinity when \( n \to +\infty \), and require this asymptotically, i.e., for \( n \) large enough. You may need the numerical estimate \( e^{-1/4} \approx 0.7788 \).)

The probability that we do \( T \) at least \( 2^{p(n)} \) times is the probability that we draw non-zero sequences of \( p(n) + 2 \) bits \( 2^{p(n)} \) times, namely \( (1 - \frac{1}{2^{p(n)+2}})^{2^{p(n)}} \). When \( n \) tends to \( +\infty \), this is equal to:

\[
(1 - \frac{1}{2^{p(n)+2}})^{2^{p(n)}} = e^{2^{p(n)} \log(1 - \frac{1}{2^{p(n)+2}})}
\]

which tends to \( e^{-1/4} \). As this is strictly larger than \( \frac{3}{4} \), we obtain the desired bound for \( n \) large enough.

10. [20 lines] Use the previous observation to produce an RL algorithm deciding whether \( t \) is reachable from \( s \) in \( G \).
Modify $A'$ so that instead of maintaining a $p(n)$-bit counter, and incrementing it each time through the outer loop, it draws $p(n) + 2$ bits independently at random, and stops if all of them are zero.

By the previous question, with probability at least $\frac{3}{4}$, the resulting algorithm $A''$ will stop after at least $2^{p(n)}$ iterations. I.e., it only has a probability at most $\frac{1}{4}$ of making an error because it stopped too early.

If $t$ is reachable from $s$, it will accept with probability $\frac{3}{4}$ times the $\frac{1}{2}$ bound from $P2\text{mod}$. If it is not reachable, then $A'$ will terminate with probability 1, and cannot accept.

It only remains to check that we can implement $A''$ using only $O(\log n)$ space. We certainly won’t draw $p(n) + 2$ bits and store them! Instead, we can iterate the following loop:

```plaintext
1 for j:=1 to $p(n) + 2$ {
2   draw one bit at random: if it is 1, then goto 5
3 } 
4 reject; // all bits were drawn as 0
5 // continue normally
```

So we obtained an $RL$ algorithm with acceptance probability $\frac{3}{4} \times \frac{1}{2} = \frac{3}{8}$. But we have observed that the acceptance probability was irrelevant provided it was strictly between 0 and 1.

11. [6 lines] Conclude that $RL = NL$.

Since $REACH$ is $NL$-complete, it suffices to observe that $RL$ is closed under logspace reductions to conclude that $NL \subseteq RL$, which concludes the proof.

That $RL$ is closed under logspace reductions is the essential argument here, and is done as for $NL$ in the lecture notes: we simulate the reduction $f$ by asking one bit of $f(w)$ at a time, simulating the machine that computes $M$ so that it throws away all the other bits of $f(w)$.