Examen de complexité avancée 2016

Printed lecture notes allowed. No access to the Internet, no computer, no cell phone.

All the theorems you use must be cited correctly, by author names; e.g., « by the Immerman-Szelepcsényi Theorem ».

At some point, you may need to use the following non-deterministic time hierarchy theorem: if \( f, g: \mathbb{N} \to \mathbb{N} \) are two functions, \( f(n) \neq 0 \) for every \( n \in \mathbb{N} \), \( g \) is monotone and time-constructible (e.g., proper), and \( f(n + 1) = o(g(n)) \), then \( \text{NTIME}(f(n)) \) is a proper subset of \( \text{NTIME}(g(n)) \).

1 Arithmetical circuits

An arithmetical circuit is defined exactly as the circuits of the lectures, except that the gates are no longer « and » and « or » gates, rather they are « plus » and « times » gates (of arbitrary fan-in—the fan-in is the number of input wires to the gate), and also « - » gates (of fan-in one). Correspondingly, we say that those gates have type « plus », « times », or « - ».

An arithmetical circuit \( C \) is said to have arity \( k \) if only if it has exactly \( k \) input wires (i.e., wires that are not connected to the output of any gate). We number those \( k \) input wires, and associate them with variables \( X_1, X_2, \ldots, X_k \) respectively.

The value of the wires of a circuit \( C \) is defined to be a polynomial in \( X_1, X_2, \ldots, X_k \) with coefficients in \( \mathbb{Z} \), by the rules:
- the input wire number \( i \) of \( C \) has value \( X_i \);
- the output wire of a « times » gate is the product of the polynomials that are the values of the input wires to the gate;
- the output wire of a « plus » gate is the sum of the values of its input wires;
- the output wire of a « - » gate is the opposite of the value of its input wire.

The value of the whole circuit is the value of its output wire. This gives you a polynomial. Let me remind you that a polynomial is a sum of monomials, encoded as a list, so \( 2X^2Y + 8XY^2 - X^2 - 4XY + X + 4Y \) is a polynomial, but \( (2XY - X + 1)(X + 4Y) \) is not.

Note that the constant 1 is encoded as a « times » gate with fan-in 0, and the constant 0 is encoded as a « plus » gate with fan-in 0.

The size of a circuit \( C \) is, as usual, the number of bits needed to represent it as a netlist, that is, as a list of lines \( \langle \text{output wire number of gate } i \rangle; \text{ type of gate } i; \langle \text{list of input wires to gate } i \rangle \). Its depth is the length of the longest path from an input wire of \( C \) to the output wire of \( C \).
1. Give an example of an arithmetical circuit of polynomial size in \( n \), even with no input wire, that defines a (constant) polynomial of exponential size.

For example, plug the \( 1 \) gate into a doubling gate (sum of its two input wires), obtaining \( 2 \). Iterating those doubling gates would only build the number \( 2^n \), of polynomial size, after \( n \) doubling gates. Instead, adding a chain of \( n \) squaring gates (product of its two input wires) builds wires with values \( 2, 4, 16, \ldots, 2^{2^n} \), of size \( O(2^n) \).

2. Give an example of an arithmetical circuit of polynomial size in \( n \), with just one input wire \( X_1 \), that defines a polynomial of exponential degree.

Using a string of \( n \) squaring gates on the input wire \( X_1 \), obtaining \( X_2^{2^n} \).

3. Define the following algorithm, which on input an arithmetical circuit \( C \) fills out a table \( d[i] \), for each wire number \( i \) in increasing order. For the input wires \( i \) of \( C \), \( d[i] \) is set to \( 1 \). For each « plus » gate, with output wire \( j \) and inputs wires \( i_1, \ldots, i_m \), set \( d[j] = \max(d[i_1], \ldots, d[i_m]) \); for each « times » gates, similarly, set \( d[j] = d[i_1] + \cdots + d[i_m] \); for each « − » gate with input \( i \) and output \( j \), set \( d[j] = d[i] \). Clearly, that algorithm computes an upper bound on the total degree of the polynomial computed by \( C \). Show that this algorithm works in polynomial time.

As in the lectures, I am assuming that the wires are sorted, so in the above paragraph, \( j > i_1, \ldots, i_m \).

The algorithm clearly does polynomially many elementary operations (additions, max), and we have to show that each works in time bounded by a fixed polynomial. In turn, this forces us to bound the size of each of the values \( d[i] \), namely to bound \( d[i] \) itself.

Whatever the case (and the worst being that of « times » gates), \( d[j] \) is less than or equal to the sum of at most \( j - 1 \) terms of the form \( d[i], i < j \). So \( d[j] \leq (j - 1)(j - 2) \cdots 2.1 = (j - 1)! \). For a circuit of size \( n \), hence with at most \( n \) gates, each \( d[i] \) is therefore bounded by \( i! \), of size \( O(i \log i) = O(n \log n) \). Hence the overall time taken is a polynomial times \( O(n \log n) \), which is again polynomial.

4. Write \( d(C) \) for the upper bound on the degree of the polynomial described by \( C \), computed by the above algorithm. Fix a natural number \( \ell \geq 1 \). Design a randomized polynomial-time algorithm that takes an arithmetical circuit \( C \) of size \( n \) as input and returns a number \( p \) of at least \( n^\ell(1 + \log_2 d(C)) \) bits as output, which is prime with probability at least \( 1 - 1/2^n \).

We draw numbers of the specified size uniformly at random until one of them is prime (which we can check by the Agrawal-Kayal-Saxena algorithm in polynomial time; or by randomized polynomial-time algorithms with some arbitrary small error). This is guaranteed to succeed in an expected polynomial time, because of Bertrand’s postulate.
Precisely, there are at least $2^N / (3(1 + N) \log 2)$ prime numbers of $N$ bits, so the probability that we find one in one iteration is at least $1 / (3(1 + N) \log 2)$. By the usual series, we will find one in at most $3(1 + N) \log 2$ iterations, where $N = n^\ell (1 + \log_2 d(C))$. That is computable, since $d$ is.

This is polynomial in $n$, since, as we have shown in the course of the previous question, $d(C)$ is at most $n^\ell$, so $\log_2 d(C) = O(n \log n)$.

To turn that into a (non-expected, rather worst-case) polynomial-time algorithm, we stop it arbitrarily after, say, $k$ steps. The error bound is of the order of $(1 - 1 / (3(1 + N) \log 2))^k$. To make that smaller than $1 / 2^n$, we require $k \log(1 - 1 / (3(1 + N) \log 2)) \leq -n^\ell$, namely $k \geq n^\ell / \log(1 - 1 / (3(1 + N) \log 2))$. Taking $k \geq n^\ell$ suffices. This is polynomial, concluding the argument.

5. Make the final step and show that the following problem PIT:

**INPUT**: an arithmetical circuit $C$

**QUESTION**: does $C$ defines the identically zero polynomial?

is in $\text{coRP}$.

Using the previous question, draw a (probably) prime number $p$. Then evaluate $C$ at random values for its input wires $X_1, \ldots, X_k$, taken uniformly from $\mathbb{Z}/p\mathbb{Z}$. (This evaluation takes polynomial time, because the size of the numbers involved is bounded by $\log_2 p$, which is polynomial in the size $n$ of $C$.)

If we obtain 0, then accept, otherwise reject.

If $C$ defines the identically zero polynomial, then we shall always accept. Otherwise, by the Schwartz-Zippel lemma, we will reject with probability at least $d(C)/p$. Since $p$ contains at least $n^\ell (1 + \log_2 d(C))$ bits, $p \geq n^\ell d(C)$, so the error is at most $1/2^n$. Summing that with the probability that $p$ might in fact not be prime, the probability of error in that case is at most $2/2^n$.

This yields a $\text{coRP}$ algorithm, with error at most $2/2^n$.

2 The permanent and arithmetic circuits

We do not know whether PIT is in $\text{P}$. Given a polynomial $p(n)$, the subproblem $p(n)$-degree-PIT (familiarly called « low-degree-PIT ») is the following:

**INPUT**: an arithmetical circuit $C$, of size $n$, with $d(C) \leq p(n)$

**QUESTION**: does $C$ defines the identically zero polynomial?

The $p(n)$-degree-PIT is of course again in $\text{coRP}$, and still not known to be in $\text{P}$.

Now consider the permanent problem. For an $m \times m$ matrix $M = (m_{ij})_{1 \leq i,j \leq m}$ with integer entries, the permanent of $M$ is defined as:

$$\text{perm } M = \sum_\sigma \prod_{i=1}^m m_{i \sigma(i)}$$  \hspace{1cm} (1)
where the sum is over all permutations $\sigma$ of $\{1, 2, \cdots, m\}$ (a permutation is a bijection from $\{1, 2, \cdots, m\}$ onto $\{1, 2, \cdots, m\}$). This is the same formula as the determinant, except with no $\pm$ sign in front of the factors.

Note that, symbolically, $\text{perm} M$ is a polynomial in $m^2$ variables $m_{ij}, 1 \leq i, j \leq m$, of total degree $m$. We shall say that an arithmetical circuit computes the permanent (of $m \times m$ matrices) if and only if the polynomial it defines is precisely the above polynomial $\text{perm}$.

Formula (1) implies that there is a depth 2 circuit that computes the permanent, of degree $m$, with $m^2$ input wires, unfortunately of exponential size. It is unknown whether the permanent has polynomial-size arithmetical circuits.

6. For all $1 \leq i, j \leq m$, the minor $M_{i,j}$ of $M$ is the matrix with row $i$ and column $j$ removed. This will be useful in connection with the following formula, which we shall admit:

$$\text{perm} M = \sum_{j=1}^{m} m_{1j} \text{perm} M_{1,j}. \quad (2)$$

In fact, for any fixed index $i, 1 \leq i \leq m$, we have the formula:

$$\text{perm} M = \sum_{j=1}^{m} m_{ij} \text{perm} M_{i,j}. \quad (3)$$

Imagine you are given an arithmetical circuit $C$ that computes the permanent $\text{perm} M$ of $m \times m$ matrices. Define a new arithmetical circuit $C'$ that computes the permanent of $(m - 1) \times (m - 1)$ matrices. (For example, $\text{perm} M_{(m-1),(m-1)}$; you may wish to compute $\text{perm} M_{i,j}$ for any other pair $i, j$; all of them are equal, up to renamings of variables).

In general, we can compute a new arithmetical circuit $C_{i,j}$ that computes $\text{perm} M_{i,j}$. Using (3), it suffices to evaluate $\text{perm} M$ with $m_{ij} = 1$ and $m_{ik} = 0$ for every $k \neq j$. Hence the new arithmetical circuit $C_{i,j}$ is just $C$, with its $m_{ij}$ entry plugged to the output of a 1 gate (a nullary « times » gate), and its $m_{ik}$ entries plugged to the output of a 0 gate (a nullary « plus » gate).

7. Using formula (2), show that the following problem $\text{PERM}$:

INPUT : an arithmetical circuit $C$ with $m^2$ input wires $m_{i,j}, 1 \leq i, j \leq m$, such that $d(C) \leq m$

QUESTION : is the polynomial defined by $C$ equal to the polynomial $\text{perm}$?

is decidable in polynomial-time using an oracle that solves $m$-degree-PIT.

We can check whether two circuits define the same polynomial by testing whether their difference is the zero polynomial. Their difference can be obtained by plugging a « − » gate at the output of one circuit, and plugging the two results into a binary « plus » gate.

It is tempting to compute an arithmetical circuit $C_{\text{perm}}$ first, and then test whether $C$ and $C_{\text{perm}}$ define the same circuit this way, but, as we have said, we do not even know whether the permanent has polynomial-size arithmetical circuits.
Instead, we use a form of self-reducibility. We build a recursive procedure as follows:

- if \( m = 0 \), then we check that \( C \) is the constant 0 circuit on no input (this can even be checked by direct computation);
- we can instead start the induction at \( m = 1 \), checking that \( C \) is the polynomial \( m_{11} \), namely that \( C + (-m_{11}) \) is the zero polynomial, using 1-degree-PIT (from now on, we equate circuits with the polynomials they define);
- otherwise, we check the following equality, analogous to (2):

\[
C = \sum_{j=1}^{m} m_{ij} C' \theta_j
\]

where \( \theta_j \) is the renaming that replaces \( m_{i'j'} \) with \( m_{(i'+1)(j'+1)} \) if \( j' \geq j \), by \( m_{(i'+1)j'} \) otherwise.

We observe that \( C' \) and \( C' \theta_j \) are computable in polynomial time (of the order of \( m^2 \) times the time to manipulate each variable, itself \( O(\log m) \)), and the equality above is checked by building the circuit

\[
C - \sum_{j=1}^{m} m_{ij} C' \theta_j
\]

and checking that it defines the zero polynomial using \( m^2 \)-degree-PIT.

8. Assume that:

- \((H_1)\) \( p(n) \)-degree-PIT is decidable in polynomial time, for any polynomial \( p \), and
- \((H_2)\) the permanent has polynomial size arithmetical circuits, say of size \( q(m) \) for \( m \times m \) matrices.

Show that \( \mathbf{P}^{\text{perm}} \subseteq \mathbf{NP} \), where \( \mathbf{P}^{\text{perm}} \) is the class of languages that are decidable in polynomial time with oracle calls that compute the permanent. (That is, our oracle calls do not just return just \( \langle \text{yes} \rangle \) or \( \langle \text{no} \rangle \), rather an integer. Formally, this would require us to change slightly the definition of an oracle Turing machine.)

Please show explicitly where you require the use of each assumption.

Let \( L \) be a language decided by a polynomial time Turing machine \( M \) making calls to an oracle computing the permanent. The dimension \( m \) of the matrices for which the oracle is queries is bounded by a polynomial \( d(n) \), in particular.

We replace each call to the oracle by the following non-deterministic procedure. We guess an arithmetical circuit of size \( q(m) \) with \( m = d(n) \). One of them will compute the permanent of \( m \times m \) matrices, by assumption \((H_2)\). We can then check that it really computes the permanent, by using the algorithm of Question 7, replacing the oracle deciding \( m^2 \)-degree-PIT by a polynomial computation, as afforded to us by assumption \((H_1)\).
9. Recall that \( \mathsf{NEXP} \) is the class of languages decidable in non-deterministic exponential time (that is, in time bounded by the exponential of some polynomial). It is known that if \( \mathsf{NEXP} \subseteq \mathsf{P}/\mathsf{poly} \), then \( \mathsf{NEXP} = \Sigma^p_2 \). It is also known, as a consequence of two celebrated theorems, one due to Valiant, the other one due to Toda, that \( \mathsf{PH} \subseteq \mathsf{P}^{\text{perm}} \). I am not asking you to prove any of those. Conclude from the previous question that, if \((H_1)\) and \((H_2)\) both hold, then \( \mathsf{NEXP} \not\subseteq \mathsf{P}/\mathsf{poly} \).

If \( \mathsf{NEXP} \subseteq \mathsf{P}/\mathsf{poly} \), then \( \mathsf{NEXP} = \Sigma^p_2 \subseteq \mathsf{PH} \subseteq \mathsf{P}^{\text{perm}} \), which is included in \( \mathsf{NP} \) by the previous question, under the hypotheses \((H_1)\) and \((H_2)\). That would imply \( \mathsf{NEXP} \subseteq \mathsf{NP} \), and we claim that this would contradict the non-deterministic time hierarchy theorem. Indeed, take \( f(n) = 2^n \), \( g(n) = 2^{2n} \). Then \( \mathsf{NEXP} \) contains \( \mathsf{NTIME}(g(n)) \), which is strictly larger than \( \mathsf{NTIME}(f(n)) \supseteq \mathsf{NP} \).

That result is due to Kabanets and Impagliazzo (2003). That is usually stated as : if low-degree-\( \mathsf{PIT} \) is decidable in polynomial time (an algorithmical question), then either the permanent does not have polynomial size arithmetical circuits or there is a non-deterministic exponential time language that does not have polynomial size circuits (which are lower bound results in complexity; both of them are widely believed to be true).

3 \( \mathsf{MA} \)

Recall that \( \mathsf{MA} \) is the class of languages \( L \) such that, for every \( \ell \geq 0 \), there is a language \( D \in \mathsf{P} \) such that, for every input \( x \), of size \( n \) :

- if \( x \in L \) then there is a \( y \) of size \( p(n) \) such that \( \Pr_r[(x, y, r) \in D] \geq 1 - 1/2^{n^\ell} \);
- if \( x \not\in L \) then for every \( y \) of size \( p(n) \), \( \Pr_r[(x, y, r) \in D] \leq 1/2^{n^\ell} \);

where the probabilities are taken over all random tapes \( r \) of size \( q(n) \), and \( p(n) \) and \( q(n) \) are two polynomials (which may depend on \( \ell \)).

10. Show that we obtain the same class by requiring no error in the \( x \in L \) case. In other words, let \( \mathsf{MA}_0 \) be the class defined as above, except for the clause :

- if \( x \in L \) then there is a \( y \) of size \( p(n) \) such that, for every \( r \) of size \( q(n) \), \( (x, y, r) \in D \).

You must show that \( \mathsf{MA} = \mathsf{MA}_0 \). As a hint, you may imitate the proof of the Sipser-Gács Theorem.

Sipser’s coding lemma will not do, since its use would require Arthur to play first and draw some hash functions at random.

Instead, use the cover-by-random-translations technique, as in the proof of the Sipser-Gács Theorem (Proposition 1.24 in the lecture notes).

Fix \( \ell \). For technical reasons, we shall use an \( \mathsf{MA} \) protocol with error \( 1/2^{n^{\ell+1}} \), not \( 1/2^{n^\ell} \).

Fix \( x \). For every \( y \) of size \( p(n) \), let \( R_y \) be the set of random tapes \( r \) such that \( (x, y, r) \in D \). If \( x \in L \), then \( R_y \) is huge (covers a proportion at least \( 1 - 1/2^{n^{\ell+1}} \).
of the whole space $\Sigma^{q(n)}$ of all random tapes). As in the proof of the Sipser-Gács Theorem, $\bigcup_{i=0}^{[q(n)/n^{\ell+1}]} (t_i \oplus R_y) = \Sigma^{q(n)}$ for some tuple of bitstrings $t_0, t_1, \ldots, t_{[q(n)/n^{\ell+1}]}$, each of size $q(n)$. In fact, this holds for at least half of all such tuples: the probability over $t_0, t_1, \ldots, t_{[q(n)/n^{\ell+1}]}$ of the complementary event, that there is an $r \in \Sigma^{q(n)}$ that is outside every $t_i \oplus R_y$, is at most $2^{q(n)}$ times $(1/2^{n^{\ell+1}}) + [q(n)/n^{\ell+1}]$, hence at most $1/2$.

In other words, we define the new one-round MA protocol as follows: on input $x$, Merlin produces $y$, Arthur draws $r$ and checks that $(x, y, r \oplus t_i) \in D$ for some $i$, $0 \leq i \leq [q(n)/n^{\ell+1}]$.

We have just seen that, if $x \in L$, then Merlin can find a $y$ such that Arthur will accept, whatever value of $r$ is drawn. If $x \notin L$, then for any answer $y$ that Merlin may give, the probability that Arthur will accept is at most $q(n) + 1$ times (one for each $i$) the probability that $(x, y, r \oplus t_i)$ is in $D$ (at most $1/2^{n^{\ell+1}}$). Asymptotically, this is at most $1/2^{n^{\ell}}$, and we conclude.

11. Conclude that MA is included in $\Sigma^p_2$.

If $L \in \text{MA} = \text{MA}_0$, then $x \in L$ if and only if there is a $y$ of size $p(n)$ such that for every $r$ of size $q(n)$, $(x, y, r) \in D$. This is clear when $x \in L$. When $x \notin L$, for every $y$ a huge proportion of random tapes $r$ (at least $1 - 1/2^{n^{\ell}}$) will falsify the claim that $(x, y, r) \in D$, hence at least one.

Now one can decide $L$ by guessing $y$ existentially, then $r$ universally, then checking $(x, y, r) \in D$ in polynomial time.