NEXP is not in ACC$^0$

Lecturer: Robert Robere  
Scribe: Robert Robere  
26 March 2014

In this lecture we discuss the recent exciting result by Williams that NEXP $\not\subseteq$ ACC$^0$. We give a sketch of the main ideas of the proof, and defer the reader to one of [5, 6, 7] for further details.

1 Preliminaries

We assume familiarity with the usual concepts of complexity theory, and in this section we review some of the more specific classes and problems that we will be studying. Recall that NEXP is the complexity class defined as

$$NEXP = \bigcup_{c\geq 0} \text{NTIME}[2^{n^c}].$$

ACC$^0$ is the class of all problems computable by constant depth, unbounded fan-in circuits with $\land, \lor, \neg$ and $\text{MOD}_m$ gates for any constant $m$. A related class to ACC$^0$ is P/poly, which contains all problems computable by polynomial-size circuits with $\land, \lor, \neg$ gates.

If $C$ is a circuit with $n$ inputs, then we use $\text{tt}(C)$ to denote the string of length $2^n$ obtained by evaluating $C$ on all $2^n$ possible inputs in lexicographic order and concatenating all of the resulting strings (the tt stands for truth table). Sometimes when we discuss circuits $C$ that take multiple inputs, we will want to hardwire some of the inputs of $C$ and leave the rest of the inputs unset. If this is the case, then we will write $C(x, \cdot)$, which denotes the circuit obtained by hardwiring the first $|x|$ inputs of $C$ using the values in $x$, and leaving the rest of the inputs unset.

The following is a canonical NEXP-Complete problem.

**Definition 1.** The Succinct 3-SAT problem is defined as follows:

**Input:** A circuit $C$ with fan-in 2 over the standard basis with $s$ gates and $n$ inputs such that $\text{tt}(C)$ encodes an exponentially large 3-SAT instance $F_C$.

**Output:** Is $F_C$ satisfiable?

Throughout this lecture, when referring to a circuit $C$ we always use $n$ to mean the number of inputs of $C$ and $s$ to mean the number of gates of $C$.

Another important problem (or, rather, family of problems) is circuit satisfiability.

**Definition 2.** Let $C$ be a class of circuits (in these notes we will take $C = \text{ACC}^0, \text{P/poly}$). The $C$ Circuit-SAT problem is defined as follows:

**Input:** A circuit $C \in C$ with $n$ inputs and $s$ gates.

**Output:** Is $C$ satisfiable?
2 Proof Overview and Outline

In this section we give a high-level overview of the proof (depicted in Figure 1). The first step is an unconditional lower bound on the running time of any nondeterministic algorithm for Succinct 3-SAT, which follows from the nondeterministic time hierarchy theorem and a very efficient 3-SAT reduction.

**Theorem 1.** Succinct 3-SAT cannot be solved in $2^{n - \omega(\log n)}$ time.

Next, assuming $\text{NEXP} \subset \text{ACC}^0$ we give an algorithm $A$ for Succinct 3-SAT which uses a deterministic algorithm for $\text{ACC}^0$ Circuit-SAT as a subroutine. When given a circuit $C$ with $n$ inputs and $s$ gates, the algorithm $A$ runs in nondeterministic time that is dominated by the best running time of the $\text{ACC}^0$ Circuit-SAT algorithm. Finally, we give a deterministic $\text{ACC}^0$ Circuit-SAT algorithm running in $2^{n - n^\varepsilon}$ time for some fixed constant $\varepsilon > 0$, contradicting Theorem 1 assuming $\text{NEXP} \subset \text{ACC}^0$.

We choose a language $L \in \text{NTIME}[2^m]$ which is not in $\text{NTIME}[o(2^m)]$, which exists by the nondeterministic time hierarchy theorem [3]. The “very efficient” reduction named before Theorem 1 from $L$ to Succinct 3-SAT follows by applying a padding argument to the next theorem:

**Theorem 2** ([1, 2]). There is a fixed constant $d$ such that for every $L \in \text{NTIME}[n]$, $L$ reduces to 3-SAT in $O(n(\log n)^d)$ time. In addition, there is an algorithm $R$ with random access to its input that given an instance of $L$ and an integer $i \in [cn(\log n)^d]$ in binary, outputs the $i$th clause of the resulting 3-SAT formula in $O((\log n)^d)$ time (where the constant $c$ depends only on the language $L$).

In the above theorem the constant $d$ depends on the underlying machine model. For example, if the machine model is a multitape Turing machine, then we can take $d = 4$ (cf. Theorem 1).

The rest of this lecture will be divided into two parts. The first part will be devoted to the construction of the algorithm $A$. As a warmup, in Section 3 we show how to construct the algorithm $A$ if we replace “$\text{ACC}^0$” with “$\text{P}/\text{poly}$”: this gives the essence of the algorithm $A$ modulo a number of complicating details introduced when considering $\text{ACC}^0$. In Section 4 we discuss some modifications to the construction in Section 3 to build an algorithm $A$ that works for $\text{ACC}^0$.

Then, in the second part of the lecture we give a “fast” $\text{ACC}^0$ Circuit-SAT algorithm (cf. Section 5). We use a wonderful result due to Beigel and Tarui [4] which gives a nontrivial “standard form” for $\text{ACC}^0$ circuits. This standard form suggests an easy divide-and-conquer algorithm (originally appearing in [5]) for $\text{ACC}^0$ Circuit-SAT that runs in $2^{n - n^\varepsilon}$ time for some fixed constant $\varepsilon$.

3 NEXP vs. P/poly

This section is devoted to the proof of the following theorem:

**Theorem 3.** If $\text{NEXP} \subset \text{P}/\text{poly}$ then there exists a nondeterministic algorithm $A$ computing Succinct 3-SAT in $t_{\text{P}/\text{poly}}(n, \text{poly}(n, s)) + \text{poly}(n, s)$ time, where $t_{\text{P}/\text{poly}}$ is the running time of any deterministic $\text{P}/\text{poly}$ Circuit-SAT algorithm.

In Section 4 we prove the same theorem when “$\text{P}/\text{poly}$” is replaced with “$\text{ACC}^0$”, but the algorithm $A$ in both cases is identical modulo some complicating details when we have to consider $\text{ACC}^0$ circuits.\footnote{In fact, a similar theorem exists if we replace “$\text{P}/\text{poly}$” with any “well-behaved” circuit class between $\text{AC}^0$ and $\text{P}/\text{poly}$, where “well-behaved” includes all of the usually considered circuit classes like $\text{NC}^i$, $\text{AC}^i$, etc.}

Before we discuss the construction, we have to briefly diverge from the path and discuss witness circuits.
3.1 Witness Circuits for NEXP

Instances of Succinct 3-SAT are circuits $C$ whose truth tables encode exponentially large 3-SAT instances. Said another way, such 3-SAT instances are “highly regular” in some way that allows us to compress their encoding as the truth table of some circuit. We can therefore rephrase the Succinct 3-SAT problem as follows: given an “exponentially compressed” representation $C$ of a 3-SAT instance $F_C$, decide whether or not $F_C$ has a satisfying assignment. This point of view suggests an interesting idea: if $F_C$ is a 3-SAT formula encoded in the truth table of a circuit $C$, could it be that the satisfying assignments of $F_C$ also have compressed representations? In other words, given a circuit $C$ for which $\text{tt}(C)$ encodes an exponentially large 3-SAT instance $F_C$, does there exist another polynomial-size circuit $W_C$ such that $\text{tt}(W_C)$ encodes a satisfying assignment to $F_C$? We define such a circuit $W_C$ to be a witness circuit for $C$ and $F_C$.

**Definition 3.** A language $L \in \text{NTIME}[f(n)]$ has $S(n)$-size witness circuits if, for every polynomial time verifier $V$ for $L$ there is an $O(S(n))$-size Boolean circuit family $\{W_{L,n}\}_{n \geq 0}$ where $W_{L,n}$ has $n + \lceil \log_2 f(n) \rceil + 1$ inputs, such that

$$\forall n \geq 0, \forall x \in \{0, 1\}^n : x \in L \iff V(x, \text{tt}(W_{L,n}(x, \cdot))) = 1.$$ 

In fact, such circuits do exist for Succinct 3-SAT if NEXP $\subseteq$ P/poly. The proof of Theorem 3 crucially relies on this fact, which is due to Impagliazzo, Kabanets and Wigderson [8].

**Theorem 4.** If NEXP $\subseteq$ P/poly then every language in NEXP has witness circuits of polynomial size.

The proof of this theorem is far beyond the scope of this lecture, but it is worth commenting that the proof is an interesting mix of the “hardness-randomness” connection and diagonalization. Williams [6] gives a self-contained treatment (Lemma 3.1 in that paper).

3.2 P/poly Circuit-SAT and NEXP

Using the witness circuits introduced in the previous section we will prove Theorem 3. The idea (at least) is simple: assuming that NEXP $\subseteq$ P/poly, we will use the polynomial size witness circuits for Succinct 3-SAT guaranteed by Theorem 4 to construct a circuit $D$ which is unsatisfiable if and only if the original Succinct 3-SAT instance is satisfiable. The algorithm $A$ then works as follows: it takes as input a circuit
(an instance of Succinct 3-SAT), and nondeterministically guesses a polynomial size witness circuit \( W_C \) for \( C \). Then, \( A \) constructs the circuit \( D \) by composing together two smaller gadgets \( G_1 \) and \( G_2 \). The first gadget, \( G_1 \), takes as input a clause number \( i \) in binary and uses copies of \( C \) to output an encoded description \( d = b_{i,1}z_{i,1}, b_{i,2}z_{i,2}, b_{i,3}z_{i,3} \) of the \( i \)th clause in \( F_C \). The variable names in the \( i \)th clause are encoded in the strings \( z_{i,1}, z_{i,2}, z_{i,3} \), and the three bits \( b_{i,1}, b_{i,2}, b_{i,3} \) flag if the corresponding variable is negated or not.

The second gadget, \( G_2 \), takes as input the description \( d \), and uses three copies of the witness circuit \( W_C \) to compute the values \( z_1^*, z_2^*, z_3^* \) of the literals \( z_1, z_2, z_3 \) under the assignment encoded by \( W_C \). Finally, \( G_2 \) outputs the negated value of the \( i \)th clause under the assignment encoded by \( W_C \). In other words, the circuit \( D \), when given a clause number \( i \), outputs 0 whenever the nondeterministically generated witness circuit \( W_C \) encodes a satisfying assignment to the \( i \)th clause. See Figure 2 for a graphical depiction.

Assuming \( \text{NEXP} \subseteq \text{P/poly} \), if \( C \) encodes a satisfiable 3-SAT formula \( F_C \) then there exists a witness circuit \( W_C \) encoding a satisfying assignment to \( F_C \). Therefore, the circuit \( D \) constructed by the nondeterministic algorithm \( A \) be unsatisfiable when using this particular witness circuit, as \( D \) will output 0 when given any clause number \( i \). Similarly, if the circuit \( D \) is unsatisfiable then the witness circuit \( W_C \) generated by the algorithm \( A \) must encode a satisfying assignment to the instance \( F_C \). If the instance \( C' \) of Succinct 3-SAT has \( n \) inputs and \( s \) gates, then constructing the circuit \( D \) takes \( O(\text{poly}(n, s)) \) time, and we can use...
a deterministic (or even co-nondeterministic) $P/poly$ Circuit-SAT algorithm to determine if the circuit $D$ is unsatisfiable. Thus the overall running time for the algorithm $A$ is $O(poly(n, s) + t_{P/poly}(n, s))$.

4 From $P/poly$ to $ACC^0$

Moving from $P/poly$ to $ACC^0$ requires quite a bit more work. Now, assuming $NEXP \subset ACC^0$, the goal is to modify the nondeterministic algorithm $A$ given in the previous section to construct another “verifier” circuit $D$ which is in $ACC^0$. The major hurdle is that both of the gadgets $G_1$ and $G_2$ constructed in the previous section are unrestricted polynomial size circuits, and we somehow have to replace them with equivalent $ACC^0$ circuits.

We leap this hurdle by using the fact that if $NEXP \subset ACC^0$, then $P \subset ACC^0$ and so there must exist an $ACC^0$ circuit family which solves the Circuit Value Problem (CVP). In fact, this observation makes replacing the gadget $G_2$ quite easy! Notice that the only components of $G_2$ that need to be replaced to make it an $ACC^0$ circuit are the witness circuits $W_C$. We can modify the algorithm $A$ to guess an $ACC^0$ circuit $V$ solving the CVP, and simply provide the description of the witness circuit $W_C$ as a hard-wired input to $V$ and use $V$ to apply $W_C$ to the variable inputs $z_{i,1}, z_{i,2}, z_{i,3}$.

![Diagram](image)

Figure 3: $V$ is an $ACC^0$ circuit computing CVP

Now we simply replace each of the copies of $W_C$ in the component $G_2$ (cf. Figure 2) with copies of the equivalent $ACC^0$ circuit shown in Figure 3, obtaining an equivalent circuit component $G'_2$ which is in $ACC^0$.

Replacing the component $G_1$ — which outputs a description of a clause in the formula $F_C$ — is a bit more difficult. The trouble is that $G_1$ has to use copies of the Succinct 3-SAT circuit $C$ to produce the clause descriptions and $C$ is a completely unrestricted circuit. A natural first attempt to scoot around this issue is to try and apply the same trick that we applied to the witness circuits $W_C$: since the Circuit Value Problem has an $ACC^0$ circuit family, we could nondeterministically guess an $ACC^0$ circuit $C'$ equivalent to $C$. But we would then have to verify that the circuits $C$ and $C'$ are equivalent, and it is not clear how to do this!

We get around equivalence verification problem by nondeterministically guessing “more information” than just an $ACC^0$ circuit $C'$ which is equivalent to $C$, and using this extra information to utilize the $ACC^0$ Circuit-SAT algorithm to verify that $C$ and $C'$ are equivalent. In particular, we construct two circuits $H, I$ computing the following functions. The circuit $I$, when given a description of a circuit $C$ and a gate index $j$ in $C$, outputs the indices of the two gates $j_1, j_2$ connected to the inputs of $j$, and a description $g \in \{\land, \lor, \neg\}$ of the gate type of gate $j$:

$$I(C, j) := (j, j_1, j_2, g)$$
It is well known (and not hard to show) that this function (the gate connection language) is computable in uniform \( \mathbf{AC}^0 \).

The second circuit \( H \) takes as input a description of a circuit \( C \), a gate index \( j \) of a gate in \( C \), and an input \( x \) to \( C \), and outputs the value of the gate \( j \) in the circuit \( C \) on input \( x \):

\[
H(C, j, x) := \text{value of gate } j \text{ in } C \text{ on input } x.
\]

The function \( H \) is computable in \( \mathbf{P} \), and so assuming that \( \mathbf{NEXP} \subset \mathbf{ACC}^0 \) it follows that there is an \( \mathbf{ACC}^0 \) circuit computing \( H \). Moreover, if \( o \) is the output gate of the circuit \( C \), then \( H(C, o, \cdot) \) is an \( \mathbf{ACC}^0 \) circuit that is equivalent to our original Succinct 3-SAT circuit \( C \).

So, we construct the \( \mathbf{ACC}^0 \) circuit \( C' = H(C, o, \cdot) \) equivalent to \( C \) as follows. First we nondeterministically guess an \( \mathbf{ACC}^0 \) circuit \( H \), defined above. To verify that \( H \) is correct, we construct another \( \mathbf{ACC}^0 \) circuit \( E \) that is unsatisfiable if and only if the \( \mathbf{ACC}^0 \) circuit \( H \) is correct. The circuit \( E \) takes as input an index of a gate \( j \) in \( C \) and a string \( x \) which is an input to the circuit \( C \). Then, \( E \) consists of two composed subcircuits. The first subcircuit is \( I \), which is an \( \mathbf{AC}^0 \) circuit computing the gate connection language of \( C \). \( E \) then feeds the gate index \( j \) into the circuit \( I \) which outputs the description \((j, j_1, j_2, g)\) of the gate \( j \). It then uses this information to check if \( H(C, j, x) = g(H(C, j_1, x), H(C, j_2, x)) \), and returns 0 if this is true. The circuit \( E \) is depicted in Figure 4; it is unsatisfiable if and only if the circuit \( H \) is correct, and we can check the unsatisfiability of \( E \) using the \( \mathbf{ACC}^0 \) Circuit-SAT algorithm in \( t_{\mathbf{ACC}}(n + \log s, \text{poly}(n, s)) \) time.

Now, some comments on how to modify the nondeterministic algorithm \( A \) discussed in Section 3.2. We first nondeterministically generate the \( \mathbf{ACC}^0 \)-equivalent witness circuit \( V_C \) (presented in Figure 3). Then, we nondeterministically generate the \( \mathbf{ACC}^0 \) circuit \( H \) described above, and construct the circuit \( E \) and use an \( \mathbf{ACC}^0 \) Circuit-SAT algorithm to verify that \( H \) is correct. Finally, we follow the construction of the circuit \( D \) in Section 3.2, using the \( \mathbf{ACC}^0 \) circuit \( V_C \) to substitute for \( V_C \) and the \( \mathbf{ACC}^0 \) circuit \( H(C, o, \cdot) \) whenever we use \( C \). The resulting modified circuit \( D' \) will be an \( \mathbf{ACC}^0 \) circuit that is unsatisfiable if and only if the 3-SAT instance \( F_C \) encoded by the original circuit \( C \) is satisfiable, and running our \( \mathbf{ACC}^0 \) Circuit-SAT algorithm on \( D' \) finishes the description. This finishes a (sketch) of the proof of the following theorem:

**Theorem 5.** If \( \mathbf{NEXP} \subset \mathbf{ACC}^0 \) then there exists a nondeterministic algorithm \( A \) running in

\[
poly(n, s) + t_{\mathbf{ACC}}(n + \log s, \text{poly}(n, s)) + t_{\mathbf{ACC}}(n, \text{poly}(s))
\]

time solving Succinct 3-SAT on circuits \( C \) with \( n \) inputs and \( s \) gates, where \( t_{\mathbf{ACC}} \) is the running time of any deterministic \( \mathbf{ACC}^0 \) Circuit-SAT algorithm.

## 5 \( \mathbf{ACC}^0 \) Circuit-SAT, Fast and Easy

In the last part of the lecture we give an \( \mathbf{ACC}^0 \) Circuit-SAT algorithm running in \( O(poly(n)2^{n−n^\varepsilon}) \) time for a fixed constant \( \varepsilon > 0 \). Combining this with Theorem 5 and Theorem 1 we conclude that \( \mathbf{NEXP} \not\subset \mathbf{ACC}^0 \).

The \( \mathbf{ACC}^0 \) Circuit-SAT algorithm relies on the following result of Beigel and Tarui [4]:

**Theorem 6.** There is an algorithm and a function \( f : \mathbb{N} \times \mathbb{N} \to \mathbb{N} \) such that given an \( \mathbf{ACC}^0 \) circuit \( C \) with \( \text{MOD}_m \) gates of \( n \) inputs, depth \( d \), and size \( s \), the algorithm outputs an equivalent depth 2 circuit structured as follows:

1. The output gate is a symmetric gate (i.e. a gate that depends only on the number of 1s given).
Figure 4: Verifying that $H(C, o, \cdot) \equiv C$
2. The symmetric gate has fan-in \(2^{\log(\ell(m,d)\ell)}\), each connected to an \(^\wedge\) gate.

3. The \(^\wedge\) gates have \(\log f(m,d)\) fan-in, and are connected to input variables.

4. None of the input variables are negated.

The algorithm runs in \(2^{\log(\log f(m,d)\ell)}\) time, and the symmetric function can be evaluated in \(2^{\log(\log f(m,d)\ell)}\) time given the number of ANDs in the circuit which evaluate to 1.

The function \(f(m,d) \leq m^{O(d)}\), and so is a large constant depending only on the input circuit. Following [5] we re-phrase this result slightly:

**Theorem 7.** There is an algorithm such that given an \(\text{ACC}^0\) circuit \(\mathcal{C}\) with \(\text{MOD}_m\) gates of \(n\) inputs, depth \(d\), and size \(s\), the algorithm outputs a symmetric function \(g : \{0,1,\ldots,K\} \to \{0,1\}\) and a multilinear polynomial \(h(x_1,x_2,\ldots,x_n)\) with \(K\) monomials such that \(C \equiv g \circ h\), where \(K = 2^{\log f(m,d)(\ell)}\). The algorithm takes at most \(O(K)\) time, and the function \(g\) can be evaluated in \(O(K)\) time.

To get Theorem 7 from Theorem 6, we set \(g\) to be the symmetric function and \(h\) to be the multilinear polynomial such that each monomial of \(h\) corresponds to the product of variables in an \(^\wedge\) gate in the circuit. Theorem 7 is useful thanks to the following lemma (we defer the proof until the end of the section):

**Lemma 1.** If \(h\) is a multilinear polynomial on \(n\) variables described by a table of the \(2^n\) coefficients of its monomials, then we can evaluate \(h\) on all \(2^n\) inputs in \(O(polyn(2^n))\) time.

Using Theorem 7 and Lemma 1 we can give a simple algorithm Eval computing \(\text{ACC}^0\) Circuit-SAT. Given an \(\text{ACC}^0\) circuit \(\mathcal{C}\) with \(n\) inputs and \(s = n^c\) gates, the first step of Eval is to arbitrarily choose \(k\) input variables \(X = \{x_1,x_2,\ldots,x_k\}\) of \(\mathcal{C}\) (we choose \(k\) later) and create a new circuit \(\mathcal{C}'\) as follows. First enumerate all \(2^k\) assignments to the variables in \(X\), and let \(C_i\) represent the circuit obtained from \(\mathcal{C}\) by hardwiring the inputs from \(X\) with the values in the \(i\)th assignment lexicographically. Then take the \(\vee\) of \(C_1,C_2,\ldots,C_{2^k}\). The result is a new \(\text{ACC}^0\) circuit \(\mathcal{C}'\) with \(s2^k\) gates and \(n - k\) inputs. Set \(k = n^{1/2f(m,d)}\), and apply Theorem 7 to the circuit \(\mathcal{C}'\) to obtain the symmetric function \(g : \{0,1,\ldots,K\} \to \{0,1\}\) and a multilinear polynomial \(h\) with \(n - k\) variables and \(K\) monomials in \(O(K)\) time, where

\[
K = 2^{\log f(m,d)(s2^k)} = 2^{\log n^{1/2\log f(m,d)(2s)}} = 2^{O(n^{1/2\log f(m,d)(2s)})}.
\]

Apply Lemma 1 to the polynomial \(h\) to obtain a table \(T\) of size at most \(2^{n-k}\) in time \(O(polyn(n-k)2^{n-k})\) time, and then linearly search through the table, evaluating \(g\) on every value. If \(g\) ever outputs 1, halt and return 1. If \(s\) is polynomial in \(n\) then the running time of this algorithm is dominated by the running time of Lemma 1, meaning that the entire algorithm runs in time \(O(2^{n-n^{1/2f(d,m)}}) = O(2^{n-n'})\) for a fixed constant \(\epsilon > 0\) depending only on \(m\) and \(d\).

**Theorem 8.** \(\text{NEXP} \not\subseteq \text{ACC}^0\).

**Proof.** If \(\text{NEXP} \subseteq \text{ACC}^0\) then by Theorem 5 there is an algorithm for Succinct 3-SAT running in non-deterministic

\[
polyn(n,s) + \tau_{\text{ACC}}(n + \log s, polyn(n,s)) + \tau_{\text{ACC}}(n, polyn(s))
\]

time. We have shown above that if \(s\) is polynomial in \(n\), then \(\tau_{\text{ACC}}(n,s) = O(2^{n-n'})\). This contradicts Theorem 1.

Finally we prove Lemma 1.
Proof of Lemma 1. Let \( h(x_1, x_2, \ldots, x_n) \) be our polynomial, and we give a straightforward divide-and-conquer algorithm for evaluating \( h \) on all \( 2^n \) inputs. If \( n = 1 \), return \( T = [h(0), h(1)] \). Otherwise, since \( h \) is multilinear we can write
\[
h(x_1, x_2, \ldots, x_n) = x_1 p_1(x_2, x_3, \ldots, x_n) + p_2(x_2, x_3, \ldots, x_n),
\]
for some multilinear polynomials \( p_1, p_2 \). Factoring \( h \) in this way takes \( 2^n \) time. Then, recursively evaluating the polynomials \( p_1, p_2 \) we obtain two tables \( T_1, T_2 \), each of size at most \( 2^{n-1} \). Given these two tables, we return
\[
T = [T_2[1], \ldots, T_2[2^{n-1}], T_1[1] + T_2[1], \ldots, T_1[2^{n-1}] + T_2[2^{n-1}]].
\]
Creating the table \( T \) takes time \( O(poly(n)2^n) \), giving a recursive running time of
\[
r(n) = 2 \cdot r(n - 1) + O(poly(n)2^n),
\]
and it is an easy exercise to see that \( r(n) = O(poly(n)2^n) \).

References


