

CMPT 710 - Complexity Theory: Week 7

Valentine Kabanets

October 12, 2006

1 PSPACE-completeness of TQBF

Theorem 1. *TQBF is PSPACE-complete.*

Proof. (1) $TQBF \in PSPACE$: we already showed that above.

(2) $TQBF$ is PSPACE-hard. We need to reduce every language in PSPACE to TQBF. According to our definition of TQBF, the inner formula must be in CNF. However, it will be easier for us to reduce every language in PSPACE to a quantified formula in DNF (disjunctive normal form). This turns out to be sufficient since PSPACE is closed under complementation. (Check this.)

Let L be any language in PSPACE. Say L is decided by some TM M in $\text{Space}(n^c)$ for some constant c . To decide if $x \in L$, we consider the configuration graph of M on x . Each configuration will be of size $O(n^c)$; let us denote this size by m . There are at most 2^m different configurations. So, $x \in L$ iff there is a path from the start configuration to the accept configuration of length at most 2^m . We will construct a QBF that will express the existence of such a path.

In a sense, our proof is a restatement of the proof of Savitch's Theorem. Recall the function $\text{Path}(a, b, i)$ that we used in the proof of Savitch's Theorem: $\text{Path}(a, b, i)$ is True iff there is a path from node a to node b of length at most 2^i . We will define a sequence of QBFs ψ_i , $i = 0, \dots, m$, where $\psi_i(A, B)$ is True iff the variables A and B encode two configurations a and b such that b is reachable from a in at most 2^i steps; note that A and B are groups of m variables each. The required QBF will then be $\psi_m(\text{Start}, \text{Accept})$, where Start and Accept are the binary strings encoding the start configuration and accept configuration, respectively.

For $i = 0$, $\psi_0(X, Y)$ is a quantified Boolean formula on free variables X and Y that is True iff either $X = Y$ or Y is reachable from X in 1 step. The formula $\psi_0(X, Y)$ can be written as a DNF of size $\text{poly}(m)$. (*Exercise:* Explain why.)

Given $\psi_i(X, Y)$, we can try to define $\psi_{i+1}(X, Y) = \exists Z[\psi_i(X, Z) \wedge \psi_i(Z, Y)]$. However, this is a *bad* idea: the size of ψ_{i+1} doubles, and so, the size of ψ_m would be exponential. The trick is to "re-use" the formula ψ_i . Here is a correct definition of $\psi_{i+1}(X, Y)$:

$$\exists Z_{i+1} \forall X_{i+1} \forall Y_{i+1} [((X_{i+1} = X \wedge Y_{i+1} = Z_{i+1}) \vee (X_{i+1} = Z_{i+1} \wedge Y_{i+1} = Y)) \Rightarrow \psi_i(X_{i+1}, Y_{i+1})]$$

Note how the formula above is True iff there is a Z_{i+1} such that both $\psi_i(X, Z_{i+1})$ and $\psi_i(Z_{i+1}, Y)$.

The rest of the proof takes care of technical details. First, note that in our definition of ψ_{i+1} all the quantifiers of the subformula ψ_i can be taken to the front, immediately after the quantifiers $\exists Z_{i+1} \forall X_{i+1} \forall Y_{i+1}$. Secondly, we need to turn ψ_{i+1} into a DNF. Since, by the inductive hypothesis, ψ_i is already a DNF, it suffices to turn into a DNF the subformula $\phi = \neg[(X_{i+1} = X \wedge Y_{i+1} = Z_{i+1}) \vee (X_{i+1} = Z_{i+1} \wedge Y_{i+1} = Y)]$. Let us denote the subformulas of ϕ by ϕ_1, \dots, ϕ_4 so that $\phi = \neg[(\phi_1 \wedge \phi_2) \vee (\phi_3 \wedge \phi_4)]$. By simple logical transformations, the equivalent DNF is

$$(\neg\phi_1 \wedge \neg\phi_3) \vee (\neg\phi_1 \wedge \neg\phi_4) \vee (\neg\phi_2 \wedge \neg\phi_3) \vee (\neg\phi_2 \wedge \neg\phi_4).$$

Each of the four subformulas can be expressed as a DNF over the variables of ϕ_i 's. For example, the first subformula $(\neg\phi_1 \wedge \neg\phi_3)$ is

$$\bigvee_{1 \leq s, t \leq m} \alpha_s \wedge \alpha_t,$$

where α_s expresses the fact that the strings X_{i+1} and X differ in position s , and α_t expresses the fact that the strings X_{i+1} and Z_{i+1} differ in position t . Finally, to express that two strings $x_1 \dots x_m$ and $y_1 \dots y_m$ differ in position s , we can use the DNF $(x_s \wedge \neg y_s) \vee (\neg x_s \wedge y_s)$.

We conclude by observing that the formula ψ_m can be constructed in logspace, since ψ_m has a very regular structure; the details are left as an exercise. \square

A few other two-person games are (e.g., Go and checkers) are also known to be PSPACE-complete (see Papadimitriou's book).

2 Polynomial-Time Hierarchy

Recall that TQBF talks about quantified Boolean formulas where the number of alternations is *non-constant*, e.g., $\exists x_1 \forall x_2 \dots Q_n x_n \phi(x_1, \dots, x_n)$ has n alternating quantifiers. Also recall that TQBF is PSPACE-complete. What happens if the number of alternating quantifiers is fixed to some constant? Then we obtain complete problems for the Polynomial-Time Hierarchy, defined below.

We call a k -ary relation R *polynomially balanced* if, for every tuple $(a_1, \dots, a_k) \in R$, the lengths of all a_i 's are polynomially related to each other.

Definition For any $i \geq 0$, a language $L \in \Sigma_i^p$ iff there is a polytime decidable, polynomially balanced $(i+1)$ -ary relation R such that

$$L = \{x \mid \exists y_1 \forall y_2 \exists y_3 \dots Q_i y_i R(x, y_1, \dots, y_i)\}.$$

Here, Q_i is \exists if i is odd, and \forall if i is even.

For example, $\Sigma_0^p = \text{P}$ and $\Sigma_1^p = \text{NP}$.

Definition For any $i \geq 0$, a language $L \in \Pi_i^p$ iff there is a polytime, polynomially balanced $(i+1)$ -ary relation R such that

$$L = \{x \mid \forall y_1 \exists y_2 \forall y_3 \dots Q_i y_i R(x, y_1, \dots, y_i)\}.$$

For example, $\Pi_0^p = \text{P}$ and $\Pi_1^p = \text{coNP}$.

Note that, in general, for every i , $\Pi_i^p = \text{co}\Sigma_i^p$.

Definition PH = $\cup_{i \geq 0} \Sigma_i^p$.

Theorem 2. $\text{PH} \subseteq \text{PSPACE}$

Proof. We know that even the general version of TQBF is in PSPACE. Hence, so is the version of TQBF with constant number of alternating quantifiers. \square

3 Examples of problems in PH

Unique-SAT = $\{\phi \mid \phi \text{ is a formula with exactly one satisfying assignment}\}$

Theorem 3. *Unique-SAT is in Σ_2^p .*

Proof. Note that $\phi \in \text{Unique-SAT}$ iff there is y such that for all z , $z \neq y$, we have $\phi(y)$ is True and $\phi(z)$ is False. \square

Min-Circuit = $\{C \mid C \text{ is a Boolean circuit s.t. no smaller equivalent circuit exists}\}$

Here, the size of a Boolean circuit is the number of logical operations (ANDs, ORs, and NOTs), or gates, used in the circuit.

Theorem 4. *Min-Circuit is in Π_2^p .*

Proof. Note that C is in Min-Circuit iff for every smaller circuit C' there is an input x such that $C(x) \neq C'(x)$. \square

4 Alternative definition of PH

Definition An *oracle TM* is a TM M with special tape, called oracle tape, and special states $q_?$, q_{yes} , q_{no} . When run with some oracle O (where O is just some language), M can query O on some strings x by writing these x onto its oracle tape, and then entering the state $q_?$. In the next step, TM M (miraculously) finds itself in the state q_{yes} if $x \in O$, or the state q_{no} if $x \notin O$.

This definition of an oracle TM captures the notion of “having access to an efficient algorithm deciding O ”.

For complexity classes C_1 and C_2 , we say that a language $L \in C_1^{C_2}$ if there is an oracle TM M from class C_1 that, given oracle access to some language $O \in C_2$, decides L .

For example, $\text{Unique-SAT} \in \text{NP}^{\text{NP}}$: Given a formula ϕ , nondeterministically guess an assignment a . Check that $\phi(a)$ is True. If not, then Reject; otherwise, construct a new formula $\phi'(x_1, \dots, x_n) \equiv \phi(x_1, \dots, x_n) \wedge [x_1 \dots x_n \neq a_1 \dots a_n]$. Ask the SAT oracle whether ϕ' is satisfiable. If it is, then Reject; otherwise, Accept. (Remark: With a more careful argument one can show that $\text{Unique-SAT} \in \text{P}^{\text{NP}}$, with only 2 queries to the SAT oracle. Do you see how?)

Alternative definition of PH. $\Sigma_0^p = \Pi_0^p = \text{P}$. For all $i \geq 0$, $\Sigma_{i+1}^p = \text{NP}^{\Sigma_i^p}$ and $\Pi_{i+1}^p = \text{coNP}^{\Sigma_i^p}$. Finally, set $\text{PH} = \cup_{i \geq 0} \Sigma_i^p$.

Theorem 5. *The original definition and the alternative definition of PH are equivalent.*

Proof. The base case of $i = 0$ is immediate: in both definitions, the 0th level is just the class P.

Just for the sake of this proof, let us denote by Σ_i^1 and by Σ_i^2 the i th level of polytime hierarchy according to definitions 1 and 2, respectively. (The first definition is in terms of logical formulas; the second definition is in terms of oracle TMs.)

We need to show that $\Sigma_i^1 = \Sigma_i^2$, for all i . The case of $i = 0$ is already argued. Let us assume the equivalence of the two definitions for i , and prove it for $i + 1$.

Let us start by proving that $\Sigma_{i+1}^1 \subseteq \Sigma_{i+1}^2$. By definition, $L \in \Sigma_{i+1}^1$ iff there is a polybalanced relation R such that $x \in L \Leftrightarrow \exists y_1 \forall y_2 \dots R(x, y_1, y_2, \dots, y_{i+1})$. Consider the language $L' = \{(x, y) \mid \forall y_2 \dots R(x, y, y_2, \dots, y_{i+1})\}$. It is easy to see that $L' \in \Pi_i^1$, and hence, by the induction hypothesis, $L' \in \Pi_i^2$. Now, to test if $x \in L$ we can do the following: Nondeterministically guess a y , then check if $(x, y) \in L'$ by querying the Π_i^2 oracle. This algorithm shows that $L \in \Sigma_{i+1}^2$.

Let us now prove the other direction, i.e., that $\Sigma_{i+1}^2 \subseteq \Sigma_{i+1}^1$. Consider an arbitrary language $L \in \Sigma_{i+1}^2$. By definition, there is an $\text{NP}^{\Sigma_i^2}$ TM M that decides L . Also, we have that $x \in L$ iff there is an accepting computation of M on x .

For any input x , consider a run of the TM M on x . During that computation, the TM M may ask (up to a polynomial number of) oracle queries to the Σ_i^2 oracle. Some of these oracle queries have the answer Yes, and the others No. Note that the Yes answers can be verified in Σ_i^2 , which is equal to Σ_i^1 , by the inductive hypothesis. The No answers can be verified in Π_i^2 , which is equal to Π_i^1 , by the inductive hypothesis.

Thus, to test if $x \in L$, we can guess (using the \exists quantifier) an accepting computation path of M on x together with all answers to the oracle queries, and check the correctness of our path, including all the answers to the oracle queries, in $(\Sigma_i^1 \cup \Pi_i^1)$. Put together, this gives us a way to check whether $x \in L$ by a Σ_{i+1}^1 formula. Hence, we get $L \in \Sigma_{i+1}^1$.

Finally, since $\Pi_i = \text{co}\Sigma_i$ for each of the two definitions, we immediately obtain the equality $\Pi_{i+1}^1 = \Pi_{i+1}^2$. \square

5 Collapsing the Polynomial-Time Hierarchy

Many results in complexity theory have the form “Statement X is true, unless the polynomial-time hierarchy collapses”. Since it is generally believed that all the levels of the polytime hierarchy are distinct, i.e., that the polytime hierarchy *does not collapse*, such results give us some evidence that the statement X is probably true.

Let us see under what conditions the polytime hierarchy would collapse to some finite level, i.e., $\text{PH} = \Sigma_i^p$, for some constant i .

Theorem 6. *If $\text{NP} = \text{coNP}$, then $\text{PH} = \Sigma_1^p = \text{NP} = \text{coNP}$.*

Proof. We want to show that $\Sigma_i^p = \Sigma_1^p$ for every i . Our proof is by induction. The base case of $i = 1$ is obvious. Suppose the truth of the statement for $i \geq 1$, and let us prove it for $i + 1$.

Take any $L \in \Sigma_{i+1}^p$. By definition, there is an $(i + 2)$ -ary polytime polybalanced relation R such that $x \in L$ iff $\exists y_1 \forall y_2 \dots R(x, y_1, \dots, y_{i+1})$. Consider the language $L' = \{(x, y) \mid \forall y_2 \exists y_3 \dots R(x, y, y_2, \dots, y_{i+1})\}$. It is clear that $L' \in \Pi_i^p$. By the inductive hypothesis, we have

$\Pi_i^p = \text{co}\Sigma_i^p = \text{co}\Sigma_1^p = \Pi_1^p$. On the other hand, our assumption is that $\text{NP} = \text{coNP}$, i.e., that $\Sigma_1^p = \Pi_1^p$. Thus, we obtain that $L' \in \Sigma_1^p$.

Finally, observe that $x \in L$ iff $\exists y (x, y) \in L'$. Since $L' \in \Sigma_1^p$, it follows that there is a polytime polybalanced relation R' such that $w \in L'$ iff $\exists z R'(w, z)$. Thus, $x \in L$ iff $\exists(y, z) R'((x, y), z)$. Therefore, $L \in \Sigma_1^p$. \square

The theorem above can be easily generalized to show the following.

Theorem 7. *If $\Sigma_i^p = \Pi_i^p$ for some $i \geq 1$, then $\text{PH} = \Sigma_i^p$.*

In other words, if the i th level of the polytime hierarchy (for $i \geq 1$) is closed under complement, then the polytime hierarchy collapses to the i th level. It is easy to see that the converse is also true.

Theorem 8. *If $\text{PH} = \Sigma_i^p$, then $\Sigma_i^p = \Pi_i^p$.*

Proof. This follows from the fact that $\text{PH} = \text{coPH}$, i.e., the polytime hierarchy is closed under complement. (Remark: Note that this is different from saying that any *fixed level* Σ_i^p is closed under complement.) For any i , we have that $\Sigma_i^p \subseteq \Pi_{i+1}^p$, and $\Pi_i^p \subseteq \Sigma_{i+1}^p$. \square

6 Circuit Complexity

6.1 Definitions

A Boolean circuit C on n inputs x_1, \dots, x_n is a directed acyclic graph (DAG) with n nodes of in-degree 0 (the inputs x_1, \dots, x_n), one node of out-degree 0 (the output), and every node of the graph except the input nodes is labeled by AND, OR, or NOT; it has in-degree 2 (for AND and OR), or 1 (for NOT). The Boolean circuit C computes a Boolean function $f(x_1, \dots, x_n)$ in the obvious way: the value of the function is equal to the value of the output gate of the circuit when the input gates are assigned the values x_1, \dots, x_n .

The *size* of a Boolean circuit C , denoted $|C|$, is defined to be the total number of nodes (gates) in the graph representation of C . The *depth* of a Boolean circuit C is defined as the length of a longest path (from an input gate to the output gate) in the graph representation of the circuit C .

A Boolean *formula* is a Boolean circuit whose graph representation is a tree.

Given a family of Boolean functions $f = \{f_n\}_{n \geq 0}$, where f_n depends on n variables, we are interested in the sizes of smallest Boolean circuits C_n computing f_n . Let $s(n)$ be a function such that $|C_n| \leq s(n)$, for all n . Then we say that the Boolean function family f is computable by Boolean circuits of size $s(n)$. If $s(n)$ is a polynomial, then we say that f is computable by polysize circuits.

It is not difficult to see (and we actually argued that in our proof of the Cook-Levin Theorem) that every language in P is computable by polysize circuits. Note that given any language L over the binary alphabet, we can define the Boolean function family $\{f_n\}_{n \geq 0}$ by setting $f_n(x_1, \dots, x_n) = 1$ iff $x_1 \dots x_n \in L$.

Is the converse true? No! Consider the following family of Boolean functions f_n , where $f_n(x_1, \dots, x_n) = 1$ iff TM M_n halts on the empty tape; here, M_n denotes the n th TM in

some standard enumeration of all TMs. Note that each f_n is a constant function, equal to 0 or 1. Thus, the family of these f_n 's is computable by linear-size Boolean circuits. However, this family of f_n 's is *not* computable by any algorithm (let alone any polytime algorithm), since the Halting Problem is undecidable. Thus, in general, the Boolean circuit model of computation is strictly more powerful than the Turing machine model of computation.

Still, it is generally believed that NP-complete languages cannot be computed by polysize circuits. Proving a superpolynomial circuit lower bound for any NP-complete language would imply that $P \neq NP$. (Check this!) In fact, this is one of the main approaches that was used in trying to show that $P \neq NP$. So far, however, nobody was able to disprove that every language in NP can be computed by linear-size Boolean circuits of logarithmic depth!

6.2 TMs that take advice

The Boolean circuit model of computation is *nonuniform*, i.e., different algorithms (circuits) are used for inputs of different sizes, and there may be no uniform algorithm (TM) that, given n , will generate the n th Boolean circuit C_n .

This uniformity can be regarded as another kind of computational resources. We can imagine a TM equipped with a special read-only tape, called *advice tape*, where some advice string $a(n)$ appears when the TM is given an input x of length n . Note that the same advice string $a(n)$ is used for all inputs of length n . If such a TM decides a language L , then we say that L is accepted by a TM with advice.

More formally, we say that $L \in \text{Time}(t(n))/f(n)$ if there is a family of advice strings $\{a_n\}_{n \geq 0}$ such that $|a_n| \leq f(n)$ for all n , and a $t(n)$ -time TM M such that, for any string x of length n ,

$$x \in L \Leftrightarrow (x, a_n) \in L(M).$$

The most important class of languages accepted by TM with advice is the class $P/\text{poly} = \cup_k \text{Time}(n^k)/n^k$.

Theorem 9. *A language L is computable by polysize Boolean circuits iff $L \in P/\text{poly}$.*

Proof Sketch. \Rightarrow . Encode a Boolean circuit C_n as an advice string a_n . Then a polytime TM with advice a_n , can decide if $x \in L$ by decoding the Boolean circuit for L from a_n and evaluating this circuit on x .

\Leftarrow . Let $\{a_n\}_{n \geq 0}$ be the family of polysize advice strings for L , and let M be a polytime TM such that $x \in L$ iff $(x, a_{|x|}) \in L(M)$. We know that every polytime decidable language is computable by polysize Boolean circuits. Let $\{C_n\}_{n \geq 0}$ be a polysize circuit family computing the language $L(M)$. We have that $x \in L$ iff $C_m(x, a_{|x|}) = 1$, where $m = |x| + |a_{|x|}| \in \text{poly}(|x|)$. By hardwiring the advice string a_n into a corresponding circuit C_m , we obtain a circuit family deciding the language L . The sizes of all these new circuits are bounded by some polynomial. \square

6.3 $NP \stackrel{?}{\subset} P/\text{poly}$

There is an interesting connection between NP having polysize circuits and the collapse of a polytime hierarchy.

Theorem 10 (Karp-Lipton). *If $NP \subset P/\text{poly}$, then $PH = \Sigma_2^P$.*

Proof Sketch. As we argued earlier, to show that $PH = \Sigma_2^P$, it suffices to prove that $\Sigma_2^P = \Pi_2^P$. To prove the latter, it actually suffices to argue that $\Pi_2^P \subseteq \Sigma_2^P$. (Show that this is indeed sufficient!)

Since $NP \subset P/\text{poly}$, there is a polysize family of circuits computing SAT. Moreover, since there is a polytime algorithm for finding a satisfying assignment, given access to an algorithm for SAT, we conclude that there is a polysize family of circuits C_n with the following property: Given a propositional formula ϕ of size n , $C_n(\phi)$ outputs a satisfying assignment for ϕ if one exists, or outputs the string of 0s if ϕ is unsatisfiable.

Now, consider any language $L \in \Pi_2^P$. By definition, there is a polytime polybalanced relation R such that $x \in L$ iff $\forall y \exists z R(x, y, z)$. Consider the language $L' = \{(x, y) \mid \exists z R(x, y, z)\}$. Obviously, $L' \in NP$. By our assumption, there is a polysize circuit family such that $C_m(x, y)$ outputs a satisfying assignment z for $R(x, y, z)$ if one exists; here, $m = |x| + |y|$.

We can test if $x \in L$ by the following polytime polybalanced formula: $\exists C_m \forall y R(x, y, C_m(x, y))$. Indeed, if $x \in L$, then there will be a polysize circuit C_m that would produce a satisfying z -assignment for $R(x, y, z)$ for every y . Conversely, if there is some small circuit C_m that produces a satisfying z -assignment for $R(x, y, z)$ for every y , then the formula $\forall y \exists z R(x, y, z)$ must be true, and hence, $x \in L$. Thus, we have shown that $L \in \Sigma_2^P$, as required. \square

6.4 Hard Boolean functions

Every Boolean function f on n variables is computable by a Boolean circuit of size $O(n2^n)$: consider a DNF formula, which is an OR of at most 2^n ANDs, where each AND is a conjunction of n literals for each x such that $f(x) = 1$. A more careful argument shows that every Boolean function on n variables is computable by a Boolean circuit of size $\frac{2^n}{n}(1 + o(1))$.

A simple counting argument shows that almost all Boolean functions are hard in the sense that they require Boolean circuits of size $\Omega(2^n/n)$. The total number of n -variable Boolean functions is $B(n) = 2^{2^n}$. On the other hand, the total number of Boolean circuits of size s on n variables is at most (very roughly) $C(n, s) = ((n+3)s^2)^s$; there are $n+3$ gate types; each gate has at most two inputs; there are s gates. When $s < 2^n/cn$, we have $C(n, s) \ll B(n)$. So, most functions require $\Omega(2^n/n)$ circuit size. Similar argument for formulas shows that most n -variable Boolean functions require $\Omega(2^n/\log n)$ formula size. Remark: more careful arguments give $2^n/n$ and $2^n/\log n$ lower bounds for circuits and formulas, respectively.

Thus, we know that hard Boolean functions abound, but we cannot get our hands on any particular hard function. That is, we do not know whether any language in NEXP requires superpolynomial circuit size.