CSE531: Computational Complexity Theory January 24, 2012

Lecture 6 **PSPACE**

Lecturer: Anup Rao

1 A complete Problem for PSPACE

The TQBF function maps the set of totally quantified boolean formulas to 0 or 1. A totally quantified boolean formula is something that looks like this:

$$\psi = \exists x_1 \forall x_2 \exists x_3 \cdots \exists x_n \phi(x_1, \ldots, x_n),$$

where here ϕ is a boolean formula on the variables $x_1, \ldots, x_n$. $\mathsf{TQBF}(\psi) = 1$ if and only ψ is true.

Theorem 1. *For every boolean $f \in$ **PSPACE**, there is a polynomial time computable function g mapping bits to truly quantified boolean formulas such that $f(x) = \mathsf{TQBF}(g(x))$.*

Proof We shall show how to use the formula to encode connectivity in the configuration graph of the machine that computes f. This is a graph of size $2^t = 2^{\mathrm{poly}(n)}$.

We generate a formula $\psi_i(A, B)$ in $\mathrm{poly}(n)$ time that checks whether there is a path of length $\leq 2^i$ from A to B. When $i = 0$. $\psi_i(A, B)$ just needs to check that B is the configuration that comes after A. Since we know that there is a polynomial sized circuit $\mathcal{C}$ such that $\mathcal{C}(x, A)$ computes the configuration that follows from A, we can construct a circuit $\mathcal{F}$ of size $\mathrm{poly}(n)$ such that

$$\mathcal{F}(A, B, x) = \begin{cases} 1 & \text{if } \mathcal{C}(A, x) = B, \\ 0 & \text{else.} \end{cases}$$

Just like in the proof that SAT is **NP**-complete, we can generate a polynomial sized formula $F(y)$ such that $\exists y F(y)$ is true if and only if $\mathcal{F}(A, B, x) = 1$.

For the general case, note that there is a path of length at most 2^i from A to B if and only if there is some vertex C in the graph such that there is a path of length 2^{i-1} from A to C and a path of length 2^{i-1} from C to A. Thus we can define

$$\psi_i(A, B) = \exists C, \psi_{i-1}(A, C) \wedge \psi_{i-1}(C, B).$$

However, this doubles the size of the formula ψ_{i-1} (which means that after t steps we will be trying to generate a formula that is exponentially big and this is impossible in polynomial time).

Indeed, we haven't yet used the $\forall$ quantifiers. Let us use the same idea as before to define the smaller formula:

$$\begin{aligned} \psi_i(A, B) &= \exists C, \forall X, \forall Y, (X = A \wedge Y = C) \vee (X = C \wedge Y = B) \Rightarrow \psi_{i-1}(X, Y) \\ &= \exists C, \forall X, \forall Y, (\neg(X = A \wedge Y = C) \wedge \neg((X = C \wedge Y = B)) \vee \psi_{i-1}(X, Y) \end{aligned}$$

The end result is a formula of size $\mathrm{poly}(n, t)$ that checks for a path of length 2^t in the graph as required. ■

2 SAT has no linear time, logspace algorithm

Although we cannot say anything non-trivial about the running time required to compute SAT, or the space required to compute SAT, we can show that SAT cannot have an algorithm that is both linear time and log space:

Theorem 2. *There is no turing machine computing* SAT *in $O(n)$ time and $O(\log n)$ space.*

In order to prove the theorem, we shall rely on two facts that we have convinced ourselves of before:

Theorem 3. *Any $f \in \mathsf{NTIME}(t(n))$ can be reduced in in logarithmic space to computing* SAT *on a formula of size $O(t(n) \log t(n))$.*

Earlier in the course we proved that the reduction is in polynomial time, but in fact it is even in **L**. (Think about this!).

Theorem 4. *If $t(n), r(n)$ are time constructible functions such that $t(n+1) = o(r(n))$, then $\mathsf{NTIME}(t(n)) \subsetneq \mathsf{NTIME}(r(n))$.*

Proof of Theorem 2: The idea is to use the purported SAT algorithm to get an unreasonable speed up of non-deterministic computations. Suppose for the sake of contradiction that SAT can be computed in linear time and logarithmic space.

Suppose $f \in \mathsf{NTIME}(n^2)$ via the non-deterministic machine M_f. We shall show how to compute f significantly faster.

By appealing to Theorem 3, consider the machine M that runs as follows on input $x \in \{0,1\}^n$:

1. Generate the formula ϕ of size $n^2 \log n$ that simulates the machine $M_f(x)$.
2. Check whether $M_f(x)$ accepts by computing $\mathsf{SAT}(\phi)$ in time $O(n^2 \log n)$ and space $O(\log(n^2 \log n)) = O(\log n)$.

M is not our final simulation. M computes f in time $O(n^2 \log n)$ and space $O(\log n)$.

Consider the configuration graph of M. This graph accepts if and only if there is an accepting path of length $t = n^2 \log n$, which happens if and only if there exist $\sqrt{t}$ intermediate configurations $C_1, \ldots, C_{\sqrt{t}}$, such that there is a path of length $\sqrt{t}$ between intermediate configurations. In other words, M accepts if and only if

$$\exists C_1, \ldots, C_{\sqrt{t}}, \forall i, C_i \text{ follows from } C_{i-1} \text{ in } \sqrt{t} \text{ steps.}$$

Each configuration takes only $O(\log n)$ bits to write down. So once we guess all of these $\sqrt{t}$ configurations, the problem of determining whether they determine an accepting of path of length t can be encoded using a SAT formula of size $O(\sqrt{t} \cdot \log n \cdot \mathsf{polylog}(t,n))$ (by Theorem 3), so it can be solved in deterministic time $O(\sqrt{t} \cdot \mathsf{polylog}(t,n))$. Thus, overall, we get a simulation in non-deterministic time $O(\sqrt{t} \cdot \mathsf{polylog}(t,n)) = O(n\mathsf{polylog}(n))$, contradicting the non-deterministic time hierarchy theorem[1] (Theorem 4) . ■

[1] Actually, we can apply the algorithm for SAT again to remove the non-determinism completely...