Lecture 11: TQBF, Randomized Computation

Anup Rao

April 30, 2024

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 $x_1, \ldots, x_n$.

$$\mathsf{TQBF}(\psi) = 1 \text{ if and only } \psi \text{ is true.}$$

TQBF's help characterize **PSPACE**.

Lemma 1. *$\mathsf{TQBF}(\psi)$ for $\psi = \exists x_1 \forall x_2 \exists x_3 \cdots \exists x_n \phi(x_1, \ldots, x_n)$ can be computed in space $O(m \cdot n)$, where size of ϕ is m. In other words, $\mathsf{TQBF} \in$ **PSPACE**.*

Proof First note that for every fixing of $x_1, \ldots, x_n$, ϕ can be computed in space $O(m)$. Let

$$A = \forall x_2 \exists x_3 \cdots \exists x_n \phi(0, x_2, \ldots, x_n)$$

and

$$B = \forall x_2 \exists x_3 \cdots \exists x_n \phi(1, x_2, \ldots, x_n).$$

We know that $\mathsf{TQBF}(\psi) = \mathsf{TQBF}(A) \vee \mathsf{TQBF}(B)$ (similarly we will have to compute $A \wedge B$ when the first quantifier is a $\forall$). Writing down A takes at most $O(m)$ space. Let $S(n)$ denote the space required to compute $\mathsf{TQBF}(\psi)$. Now, computing A recursively uses $S(n-1)$ space. After computing A, we can store the answer (one bit) and erase all contents of the tape that was used to compute A. We then write down B and compute $\mathsf{TQBF}(B)$ recursively. Overall, we have that $S(n) = S(n-1) + O(m)$. As we know that $S(0) = O(m)$, we can conclude that $S(n) = O(m \cdot n)$. ■

Theorem 2. *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 $\text{poly}(n)$ 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 $\text{poly}(n)$ 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 $\text{poly}(n, t)$ that checks for a path of length 2^t in the graph as required. ■

Lower Bounds on SAT

The material in this section was not discussed in class. We include it here as you might find it interesting. 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 3. *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 4. *If $t(n) \geq \Omega(n)$, any $f \in \mathsf{NTIME}(t(n))$ can be reduced in in logarithmic space and time $O(t(n)\log(t(n)))$ 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!). The reduction works by first computing a circuit that simulates the computation of a machine, and then computing the formula that simulates the execution of the circuit.

Another theorem we shall appeal to is the deterministic time hierarchy theorem:

Theorem 5 (Time Hierarchy)**.** *If r, t are time-constructible functions satisfying $r(n)\log r(n) = o(t(n))$, then $\mathsf{DTIME}(r(n)) \subsetneq \mathsf{DTIME}(t(n))$.*

Proof of Theorem 3: Assume for the purpose of contradiction that there is a turing machine computing SAT in $O(n)$ time and $O(\log n)$ space. The idea is to use the purported SAT algorithm to get an unreasonable speed up of computations. Suppose for the sake of contradiction that SAT can be computed in linear time and logarithmic space.

Suppose that $f \in \mathsf{DTIME}(n^2)$ via the machine M_f and $f \notin \mathsf{DTIME}(n\mathsf{polylog}(n))$. Such an f exists by Theorem 5. We shall show how to compute f in time $O(n\mathsf{polylog}(n))$, giving us the desired contradiction.

By appealing to Theorem 4, 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 $M_f(x)$, using Theorem 4.
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 $t = O(n^2 \log n)$, 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, $f(x) = 1$ 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 4), so it can be solved in deterministic time $O(\sqrt{t} \cdot \mathsf{polylog}(t, n))$. Thus we can compute a formula ψ of size $O(\sqrt{t} \cdot \mathsf{polylog}(t, n))$ such that $f(x) = 1$ if and only if

$$\exists C_1, \ldots, C_{\sqrt{t}}, \exists z, \psi(C_1, \ldots, C_{\sqrt{t}}, z).$$

The above is an instance of SAT and can then be solved deterministically in time $O(\sqrt{t} \cdot \mathsf{polylog}(n, t))$. Thus, overall, we get a simulation in deterministic time $O(\sqrt{t} \cdot \mathsf{polylog}(t, n)) = O(n\mathsf{polylog}(n)) = o(n^2/\log n)$, contradicting the deterministic time hierarchy theorem. ■