CS 6810: Theory of Computing **Spring 2026**

Lecture 7: February 12, 2026

Lecturer: Eshan Chattopadhyay *Scribe: Saket Murali*

1 TQBF - True Quantified Boolean Formula

1.1 Definition and Basic Properties

In order to continue our discussion of PSPACE and its properties, we study the language TQBF, or a true quantified boolean formula.

Definition 1.1. *A quantified Boolean formula has the form* $\phi = Q_1 x_1 \, Q_2 x_2 \ldots Q_n x_n \, \psi(x_1, \ldots, x_n)$ *where* $\psi(\cdot)$ *is a boolean formula and each* $Q_i \in \{\forall, \exists\}$*. The formula has* m *clauses, and the total number of variables is* n*.*

Remark 1.2. *If all quantifiers* Q_i *are* $\exists$*, then* TQBF *reduces to* SAT *(which is* NP*-complete). If all quantifiers* Q_i *are* $\forall$*, then* TQBF *reduces to* TAUTOLOGY *(which is co-*NP*-complete).*

More generally, we can capture the entire polynomial hierarchy using quantified boolean formulas. A formula with $i-1$ alternations of quantifiers, starting with $\exists$, corresponds to Σ_i-SAT (complete for Σ_i^P). Similarly, a formula with $i-1$ alternations starting with $\forall$ corresponds to Π_i-SAT (complete for Π_i^P). Thus, TQBF with alternating quantifiers captures the complete problems for each level of the polynomial hierarchy.

2 Proving TQBF is PSPACE-complete

Throughout this section, we will prove the following claim about TQBF and how it encompasses PSPACE.

Claim 2.1. TQBF *is* PSPACE*-complete, which means:*

1. TQBF $\in$ PSPACE
2. *For any* $L \in$ PSPACE*,* $L \leq_p$ TQBF *(i.e.,* L *is polynomial-time Karp reducible to* TQBF*)*

2.1 Part 1: TQBF $\in$ PSPACE

Definition 2.2. *Let* $S(n, m)$ *denote the space used to evaluate a quantified Boolean formula with* n *variables and description size* m*.*

We now prove the first part of **Claim 2.1** by showing that TQBF can be decided in polynomial space.

Proof. Let $\phi = Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \psi(x_1, x_2, \ldots, x_n)$ be a quantified Boolean formula with n variables, where $|\psi| = m$ denotes the size of ψ. We present a recursive algorithm A that decides the truth of ϕ in polynomial space.

Base Case: If $n = 0$ (no variables remain), then the formula contains only constants and can be evaluated in $O(m)$ space.

Recursive Case: Let $n > 0$ and let ϕ be as above. For $b \in \{0, 1\}$, denote by $\phi_{x_1=b}$ the modification of ϕ where the first quantifier Q_1 is dropped and all occurrences of x_1 are replaced with the constant b.

Algorithm A works as follows:

- If $Q_1 = \exists$: We need at least one assignment to make the formula true. Initialize an output bit to 0 (false). Recursively compute $A(\phi_{x_1=0})$. If it returns 1, output 1 and halt. Otherwise, *reuse the same space* to compute $A(\phi_{x_1=1})$ and output its result.
- If $Q_1 = \forall$: We need both assignments to make the formula true. Initialize an output bit to 1 (true). Recursively compute $A(\phi_{x_1=0})$. If it returns 0, output 0 and halt (since the formula is false for some value). Otherwise, *reuse the same space* to compute $A(\phi_{x_1=1})$ and output its result.

Space Analysis: The key insight is that we can *reuse space* between recursive calls. After computing $A(\phi_{x_1=0})$, algorithm A only needs to retain a single output bit from that computation, and can reuse the rest of the space for computing $A(\phi_{x_1=1})$.

As $S(n, m)$ was defined above, we can derive a recurrence for the amount of space used:

$$S(n, m) \leq S(n - 1, m) + O(m)$$

The intuition behind this formula is that after processing x_1, we recurse on a formula with $n - 1$ variables. The number of quantifiers decreases by 1, but the formula size remains $O(m)$ since we're only substituting a constant. Additionally, we need $O(m)$ space to write down the modified formula $\phi_{x_1=b}$ for the recursive call.

Solving this recurrence:

$$\begin{aligned} S(n, m) &\leq S(n - 1, m) + O(m) \\ &\leq S(n - 2, m) + O(m) + O(m) \\ &\leq S(n - 3, m) + 3 \cdot O(m) \\ &\vdots \\ &\leq S(0, m) + n \cdot O(m) \\ &= O(m) + O(n \cdot m) \\ &= O(n \cdot m) \end{aligned}$$

Therefore, $S(n, m) = O(n \cdot m)$, which is polynomial in the input size. Thus, TQBF $\in$ PSPACE.

Remark 2.3. *Note that this algorithm works for* TQBF *formulas with an arbitrary number of quantifiers n. This is crucial because the i-th level of the polynomial hierarchy (Σ_i^P or Π_i^P) only allows a fixed constant number of quantifier alternations (specifically, $i-1$ alternations). Since our algorithm handles formulas where n can grow with the input size,* TQBF *captures not just a single level of the polynomial hierarchy, but the entire hierarchy, placing it in* PSPACE.

□

2.2 Part 2: For any $L \in$ PSPACE, $L \leq_p$ TQBF

We now show the second part of Claim 2.1: every language in PSPACE can be reduced to TQBF in polynomial time.

Proof. Let L be a language $\in$ PSPACE and let M be a Turing machine that decides L in $S(n)$ space. For an input $x \in \{0,1\}^n$, we construct a quantified Boolean formula ϕ of size $O(S(n)^2)$ such that $x \in L$ if and only if $\phi \in$ TQBF.

Configuration Graph Approach: Recall that we can construct a configuration graph $G_{M,x}$ where:

- Each node represents a configuration of M on input x
- Each configuration can be encoded using $m = O(S(n))$ bits
- There is an edge (C, C') if configuration C can transition to configuration C' in one step
- M accepts x if and only if there is a path from C_{start} to C_{accept} in $G_{M,x}$

The challenge is that $G_{M,x}$ has up to $2^{O(S(n))}$ nodes, so we cannot write down the entire graph in polynomial time. Instead, we use a recursive quantified formula.

Recursive Formula Construction:

Definition 2.4. *For $i \geq 0$, let $Q_i(C, C')$ be a quantified Boolean formula that is true if and only if there exists a path of length at most 2^i from configuration C to configuration C' in $G_{M,x}$.*

Our goal is to construct $\phi = Q_{cS(n)}(C_{\text{start}}, C_{\text{accept}})$ for some constant c, since the configuration graph has at most $2^{cS(n)}$ nodes.

Base Case ($i = 0$): We can construct a polynomial-size Boolean formula $\varphi_M(C, C')$ that is true if and only if $(C, C') \in E(G_{M,x})$, i.e., configuration C can transition to C' in one step. This formula checks whether the two configurations differ only in the positions adjacent to the tape head and that the transition follows M's transition function. Thus:

$$Q_0(C, C') = \varphi_M(C, C')$$

Inductive Step: Suppose we have defined Q_{i-1} for some $i \geq 1$. The key observation is: there is a path of length at most 2^i from C to C' if and only if there exists an intermediate configuration D such that there are paths of length at most 2^{i-1} from C to D and from D to C'.

Naive (incorrect) approach:

$$Q_i(C, C') = \exists D \left[Q_{i-1}(C, D) \wedge Q_{i-1}(D, C') \right]$$

This approach fails because it leads to exponential formula size. If we let $f(i)$ denote the size of Q_i, then:

$$f(i) = 2f(i-1) + O(S(n)) \implies f(cS(n)) = 2^{O(S(n))}$$

which is exponential, not polynomial! Additionally, this uses only existential quantifiers, which would incorrectly reduce PSPACE to NP.

Correct approach using alternating quantifiers: Instead, we use universal quantifiers to avoid duplication:

$$Q_i(C, C') = \exists D \, \forall E \, \forall F \left[(E = C \wedge F = D) \vee (E = D \wedge F = C') \right] \implies Q_{i-1}(E, F)$$

The intuition is that we use the universally quantified variables E and F to "route" both recursive calls through a single copy of Q_{i-1}. The formula states: there exists an intermediate configuration D such that for all choices of (E, F), if (E, F) is either (C, D) or (D, C'), then $Q_{i-1}(E, F)$ holds.

Size Analysis: With this definition, we have:

$$f(i) = f(i-1) + O(m)$$

where $m = O(S(n))$ is the number of bits needed to encode a configuration. This is because we only include one copy of Q_{i-1} plus $O(m)$ additional overhead for the quantifiers and Boolean operations (the variables D, E, F, and the equality/implication checks).

Therefore, solving this recurrence:

$$f(cS(n)) = f(0) + cS(n) \cdot O(m) = O(m) + O(S(n)) \cdot O(S(n)) = O(S(n)^2)$$

Thus, the final formula $Q_{cS(n)}(C_{\text{start}}, C_{\text{accept}})$ has size $O(S(n)^2)$, which is polynomial in the input size.

Conclusion: The reduction $f : x \mapsto Q_{cS(n)}(C_{\text{start}}, C_{\text{accept}})$ can be computed in polynomial time, and $x \in L$ if and only if $f(x) \in \mathsf{TQBF}$. Thus, $L \leq_p \mathsf{TQBF}$. □

Since $\mathsf{TQBF} \in \mathsf{PSPACE}$ (Part 1) and every language in PSPACE reduces to TQBF (Part 2), we conclude that TQBF is PSPACE-complete.

3 Savitch's Theorem

Note that the proof presented above did not require the configuration graph to have out-degree equal to one. So, a similar proof structure shown below can work for nondeterministic space classes as well.

Theorem 3.1 (Savitch's Theorem [2])**.** *For any space-constructible function $S : \mathbb{N} \to \mathbb{N}$ with $S(n) \geq \log n$,*

$$\mathsf{NSPACE}(S(n)) \subseteq \mathsf{DSPACE}(S(n)^2)$$

Corollary 3.2. $\mathsf{PSPACE} = \mathsf{NSPACE}$

Proof of Savitch's Theorem. The proof closely follows our proof that TQBF is PSPACE-complete. Let $L \in \mathsf{NSPACE}(S(n))$ be a language decided by a nondeterministic Turing machine M. For input $x \in \{0,1\}^n$, the configuration graph $G = G_{M,x}$ has at most $N = 2^{O(S(n))}$ vertices, where each configuration can be encoded using $O(S(n))$ bits.

We describe a deterministic recursive algorithm $\mathsf{REACH}(u, v, i)$ that returns 1 if there exists a path from configuration u to configuration v of length at most 2^i, and 0 otherwise.

Goal: We want to compute $\mathsf{REACH}(C_{\text{start}}, C_{\text{accept}}, c \cdot S(n))$ for some constant c, since there are at most $2^{cS(n)} = N$ configurations.

Base Case ($i = 0$): We need to check if there is a path of length at most $2^0 = 1$ from u to v. This is true if and only if $u = v$ or there is a direct edge from u to v in $G_{M,x}$.

Since each configuration can be encoded in $O(S(n))$ bits, we can check whether $u = v$ or whether u can transition to v in one step according to M's transition function in $O(S(n))$ space. Thus:

$$\mathsf{REACH}(u, v, 0) = \begin{cases} 1 & \text{if } u = v \text{ or } (u, v) \in E(G_{M,x}) \\ 0 & \text{otherwise} \end{cases}$$

Recursive Case ($i > 0$): The key observation is: there exists a path of length at most 2^i from u to v if and only if there exists an intermediate configuration w such that there are paths of length at most 2^{i-1} from u to w and from w to v.

The algorithm enumerates over all possible intermediate configurations w and checks:

- Does $\mathsf{REACH}(u, w, i-1) = 1$?
- Does $\mathsf{REACH}(w, v, i-1) = 1$?

If both conditions hold for some w, output 1. Otherwise, output 0.

The crucial observation is that we can *reuse space* between the two recursive calls. After computing $\mathsf{REACH}(u, w, i-1)$, we only need to store its 1-bit output, then reuse the same space to compute $\mathsf{REACH}(w, v, i-1)$.

Space Analysis: Let $f(i)$ denote the space used by $\mathsf{REACH}(u, v, i)$. The algorithm needs:

- $O(\log N) = O(S(n))$ space to store the intermediate configuration w
- $f(i-1)$ space for the recursive calls (reused)
- $O(1)$ space to store the output of the first recursive call

This gives the recurrence:

$$f(i) = f(i-1) + O(\log N) + O(1) = f(i-1) + O(S(n))$$

We need to evaluate this at $i = \log N$ (since paths can have length at most $N = 2^{O(S(n))}$). Solving:

$$\begin{aligned} f(\log N) &= f(0) + \log N \cdot O(S(n)) \\ &= O(S(n)) + O(S(n)) \cdot O(S(n)) \\ &= O(S(n)^2) \end{aligned}$$

Therefore, $\mathsf{NSPACE}(\mathrm{S(n)}) \subseteq \mathsf{DSPACE}(S(n)^2)$, and it follows that $\mathsf{PSPACE} = \mathsf{NSPACE}$. □

References

[1] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. 1st. USA: Cambridge University Press, 2009.

[2] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.