1 TQBF and PSpace

To finish what we left last time, recall:

Name: TQBF

Input: An integer $k$ and a Boolean formula $\exists X_1, \forall X_2, \ldots, (\exists/\forall) X_k \varphi(X_1, \ldots, X_k)$ where \( \varphi \) is quantifier-free.

Output: Is the formula valid?

The difference between TQBF and QBF$_k$ are that there is no fixed level of quantifier alternations in TQBF.

**Theorem 1.** TQBF is PSpace-complete.

**Proof.** One direction is easy, namely TQBF $\in$ PSpace. Once again, to achieve a space-efficient algorithm, we use recursion. If the leading quantifier is $\exists x$, then we recursively check the two cases of setting $x$ to 0 and 1, and return true if one of them is true. Similarly, if the leading quantifier is $\forall x$, then we recursively check the two cases of setting $x$ to 0 and 1, and return true if both of them are true. At any point of the recursion, we will only need polynomial space. The recursion depth is $n$, and therefore this is a polynomial space algorithm.

For the other direction, let $M$ be a TM with space bound $p(n)$ and $x$ be an input. Recall that $M$ accepts $x$ if and only if there is an accepting path from $q_0$ to $q_{acc}$ in the configuration graph $G_{M,x}$, whose number of vertices is $2^{p(n)}$ for some constant $c$. Next we express this property by a TQBF $\varphi$.

The basic idea is the same as Savitch’s theorem. To encode that $q_1$ can reach $q_2$ in $2^\ell$ steps, denoted $q_1 \rightarrow_{2^\ell} q_2$, we go through all possible middle points $q'$. Namely we ask whether $\exists q'(q_1 \rightarrow_{2^{\ell-1}} q') \land (q' \rightarrow_{2^{\ell-1}} q_2)$. Now, notice that if we recursively expand the $\rightarrow$ inside, we would end up with an exponential size formula. The trick, is to rewrite $(q_1 \rightarrow_{2^{\ell-1}} q') \land (q' \rightarrow_{2^{\ell-1}} q_2)$ as

$$\forall x, y \quad ((x = q_1 \text{ and } y = q') \lor (x = q' \text{ and } y = q_2)) \Rightarrow (x \rightarrow_{2^{\ell-1}} y).$$

Basically, we trade one $\rightarrow$ with a $\forall$ quantifier and a couple of new variables. Now, we may recursively expand $\rightarrow$ inside the expression. Eventually we end up with a TQBF whose length is $O(p(n))$. This TQBF is valid if and only if there is an accepting path in $G_{M,x}$. \( \square \)
Clearly QBF\(_k\) is a special case of TQBF for any \(k\). Hence, \(\text{PH} \subseteq \text{PSpace}\).

TQBF captures many problems in game theory. Think of odd quantifiers (all are \(\exists\)) as the strategy of player one, and even quantifiers (all are \(\forall\)) as the counter-strategy of player two, and the Boolean formula encodes the claim that “player one wins”. Then the validity of such a formula asks the existence of a winning strategy of player one.

## 2 Circuit models

So far we have been focusing on the standard (uniform) Turing machine as our model of computation. Another arguably more natural and seemingly simpler model is Boolean circuits. Boolean circuits are also equivalent to TMs that “take advices”. We will define things formally next.

A Boolean circuit \(C\) with \(n\) inputs is a directed acyclic graph (DAG), with \(n\) sources (no incoming arcs) and one sink (no outgoing arcs). The \(n\) sources are the input bits and the sink is the output. Every internal vertex is a “gate”, labelled with \(\neg\), \(\land\), or \(\lor\). The negation gate \(\neg\) has fan-in (number of incoming arcs) 1, and we will assume \(\land\) and \(\lor\) have fan-in 2.\(^1\)

The size of \(C\), denoted \(|C|\), is the number of vertices in \(C\).

Notice that we actually do not restrict the fan-out (number of outgoing arcs). In contrast, a Boolean formula is a circuit where all internal gates have fan-out 1. (A Boolean formula is essentially a tree.) The advantage of larger fan-out is that we may reuse an intermediate value more than once.

Given an input \(x = (x_1, \ldots, x_n) \in \{0, 1\}^n\), the evaluation of \(C(x)\) goes in the straightforward way. Namely, we evaluate each gate according to the topological ordering of \(C\) gradually, until the output is computed.

All Boolean functions \(f : \{0, 1\}^n \rightarrow \{0, 1\}\) can be computed by a circuit, simply by transforming its truth table into a CNF or DNF. The catch is that it may take exponential size. A function as simple as parity: \(\oplus(x) := \sum_{i=1}^n x_i \mod 2\) requires exponential size CNF or DNF.

**Definition 1.** Let \(T : \mathbb{N} \rightarrow \mathbb{N}\) be a function. A \(T(n)\)-size circuit family \(C\) is a sequence \(\{C_n\}_{n \in \mathbb{N}}\) of Boolean circuits, where \(C_n\) has \(n\) inputs, and \(|C_n| \leq T(n)\) for every \(n\).

A circuit family \(C\) defines its corresponding language:

\[
L(C) := \{x \mid C_n(x) = 1 \text{ if } |x| = n\}.
\]

For a function \(T\), the class \(\text{Size}[T]\) is defined as

\[
\text{Size}[T] := \{L \mid \exists a T(n)\text{-size circuit family } C \text{ s.t. } L = L(C)\}.
\]

We are most interested in polynomial sized circuit families.

**Definition 2.** \(P_{/\text{poly}} := \bigcup_{c \in \mathbb{N}} \text{Size}[n^c]\).

\(^1\)This restriction does not change the power of the circuit. Since we can easily replace a \(\land\) or \(\lor\) with fan-in \(k\) by \(k-1\) gates all with fan-in 2.
The circuit model is *non-uniform* in the sense that one can change the algorithm for inputs of different sizes. There are also uniform versions where we require that an algorithm exists to compute the circuits for a given input length first. Nevertheless, the non-uniform version can be argued as efficient computation (as in hardware chips handling inputs up to a certain size), and is indeed more powerful than the standard polynomial time.

**Theorem 2.** \( \text{P} \subseteq \text{P}_{/\text{poly}} \).

*Proof.* The proof of this theorem is similar to that of the Cook-Levin theorem. Suppose \( M \) is a TM with a polynomial-time bound \( P(n) \). Recall that we construct a \( P(n) \)-by-\( O(P(n)) \) computational table. Similar to the proof of the Cook-Levin theorem, we can write down Boolean formulae to verify the following things: the first row is the initial configuration, the last row is accepting, and the content of a row is computed according to its previous row. Hence, the total size of the circuit is just \( O((P(n))^2) \), which is still a polynomial. \( \Box \)

On the other hand, all unary languages are in \( \text{P}_{/\text{poly}} \). A language \( L \) is called *unary* if \( L \subseteq \{1^n \mid n \in \mathbb{N}\} \). However, the following unary language is undecidable.

**Name:** UNARY-HALTING

**Input:** \( 1^n \)

**Output:** Does \( n \) encode a pair \( \langle M, x \rangle \) such that \( M \) halts on the input \( x \)?

As \( \text{UNARY-HALTING} \in \text{P}_{/\text{poly}} \), the class \( \text{P}_{/\text{poly}} \) is far more powerful than \( \text{P} \). It is actually very hard to place \( \text{P}_{/\text{poly}} \) amongst the complexity classes we have introduced so far. People conjecture that \( \text{NP} \nsubseteq \text{P}_{/\text{poly}} \) (which would imply \( \text{P} \neq \text{NP} \)), but there is no promising route towards proving it.

### 2.1 TM taking advices

An alternative way to characterize \( \text{P}_{/\text{poly}} \) is via TMs that take advices.

**Definition 3.** For functions \( F : \mathbb{N} \to \mathbb{N} \) and \( A : \mathbb{N} \to \mathbb{N} \), the complexity class \( \text{DTime}[F]/A \) consists of languages \( L \) such that there exist a TM with time bound \( F(n) \) and a sequence \( \{a_n\}_{n \in \mathbb{N}} \) of “advices” satisfying:

- \( |a_n| \leq A(n) \);
- for \( |x| = n \), \( x \in L \) if and only if \( M(x, a_n) = 1 \).

The following theorem explains the notation \( \text{P}_{/\text{poly}} \), namely “polynomial-time with polynomial advice”.

**Theorem 3.** \( \text{P}_{/\text{poly}} = \bigcup_{c,d \in \mathbb{N}} \text{DTime}[nc]/n^d \).
Proof. If \( L \in \text{P/poly} \), then it can be computed by a family \( \mathcal{C} = \{ C_1, C_2, \cdots \} \) of Boolean circuits. Let \( a_n \) be the description of \( C_n \), and the polynomial time machine \( M \) just reads this description and simulates it. Hence \( L \in \bigcup_{c,d \in \mathbb{N}} \text{DTIME}[n^c]/n^d \).

For the other direction, if a language \( L \) can be computed in polynomial-time with polynomial advice, say by TM \( M \) with advices \( \{ a_n \} \), then we can construct circuits \( \{ D_n \} \) to simulate \( M \), as in Theorem 2. Hence, \( D_n(x, a_n) = 1 \) if and only if \( x \in L \). The final circuit \( C_n \) just does exactly what \( D_n \) does, except that \( C_n \) “hardwires” the advice \( a_n \). Namely, \( C_n(x) := D_n(x, a_n) \). Hence, \( L \in \text{P/poly} \).

### 3 Circuit lower bounds

The attempt to separate \( \text{NP} \) from \( \text{P/poly} \) initiated the study of circuit lower bounds. It is actually very easy to show that there exists hard functions. The difficulty is to find a hard function within \( \text{NP} \). We show this easy result next, and will come back to circuit lower bounds later.

**Theorem 4.** For any \( n \), there is a function \( f : \{0,1\}^n \to \{0,1\} \) that requires at least \( \frac{2^n}{Cn} \) gates to compute for some constant \( C \).

**Proof.** The proof is a simple counting argument. The total number of functions is \( 2^{2^n} \). For a circuit of size \( S \), it can be described using \( C \cdot S \log S \) bits for some constant \( C \). Hence the total number of such circuits is at most \( 2^{C \cdot S \log S} \).

Let \( S = \frac{2^n}{(C+1)n} \). Then circuits of size \( S \) can compute at most

\[
2^{C \cdot S \log S} \leq 2^{\frac{C}{C+1} \cdot \frac{2^n}{n}} < 2^n.
\]

Hence, there must be a function \( f \) that circuit up to size \( S \) cannot compute. \( \square \)

**Remark** (Bibliographic). Relevant chapters are [AB09, Chapter 6] and [Pap94, Chapter 17].

### References
