1 Recap

Definition 1 (Boolean circuit) A Boolean circuit on $n$ inputs, is a DAG with $n$ source nodes and one target (sink) node, $m$ internal nodes perform simple computation (AND, OR, NOT).

Definition 2 ($P/poly$) $P/poly \overset{\text{def}}{=} \{\text{languages } L \text{ s.t. } \forall n, L \cap \{0, 1\}^n \text{ can be computed by a circuit of size } n^{O(1)}\}.$

Theorem 3 $P \subseteq P/poly$.

2 Turing Machines with Advice

We can give another equivalent definition of $P/poly$ by introducing the TM that takes advice.

Definition 4 (TIME($T(n)$)/$a(n)$) A language $L \in$ TIME($T(n)$)/$a(n)$ if $\exists M \in$ TIME($T(n)$) and a sequence $\{\alpha_n\}_{n \in \mathbb{N}}$ with $\alpha_n \in \{0, 1\}^{a(n)}$ s.t. $x \in L \iff M(x, \alpha_{|x|}) = 1$ for any $x \in \{0, 1\}^*$.

Claim 5 $P/poly = \bigcup_{c,d \geq 1} \text{TIME}(n^c)/(n^d)$.

Proof

($\subseteq$) For any language $L \in P/poly$. $\forall n \in \mathbb{N}$, there exists a circuit $C_n$ of size $n^d$. We treat $C_n$ as an advice to the corresponding TM $M$. $M$ evaluates input $x$ on $C_n$ given as an advice (extra input). The advice sequence $\{\alpha_n\}_{n \in \mathbb{N}} \overset{\text{def}}{=} \{C_n\}_{n \in \mathbb{N}}$ is bounded by $a(n) \overset{\text{def}}{=} n^d$.

($\supseteq$) Fix a language $L \in$ TIME($n^c$)/$n^d)$. $\forall n \in \mathbb{N}$, there exists a circuit $C_n$ of size $n^k$ for some constant $k$ s.t. $x \in L \iff C_n(x, \alpha_{|x|}) = 1$ where $|x| = n$. The advice $\alpha_n$ is hard-wired to the circuit. This is illustrated in Figure 1.

![Figure 1: P/poly $\supseteq \bigcup_{c,d \geq 1} \text{TIME}(n^c)/(n^d)$](image-url)
Remark. Claim 5 explains the reason for naming P/poly: “deterministic poly-time taking poly-size advice”. Similarly, we can define P/log, P/c where c stands for constant-sized advice, etc. A natural question arises:

Question 6 What can we get from just 1 bit of advice? P/1 = \cup_{c \geq 1} \text{TIME}(n^c)/1.

Question 7 Is NP ⊆ P/poly?

This is an open question. We believe the answer is no.

3 The Karp-Lipton Theorem

Theorem 8 (Karp-Lipton Theorem [1]) If NP ⊆ P/poly then PH = Σ₂.

Remark. Note that PH collapses to Σ₂-complete instead of NP-complete. The theorem is of the style that “one thing unlikely implies another thing unlikely”. Finally we see that many unlikely things are equivalent. In some sense it’s similar to the NP-complete problems. It has a sister-theorem, the Meyer’s Theorem, also proved in early 1980’s.

Theorem 9 (Meyer’s Theorem [1]) If EXP ⊆ P/poly then EXP = Σ₂.

Proof of Theorem 8: Suppose 3-SAT ∈ P/poly. So \( \forall n \in \mathbb{N} \), there exists a circuit \( C_n \):

\[
C_n(\phi) = \begin{cases} 
1 & \text{If } \phi \text{ satisfiable} \\
0 & \text{Otherwise}
\end{cases}
\]

where \( \phi \) is a 3-CNF on \( n \) variables. Also, \( |C_n| = \text{poly}(n) \).

Algorithm: Input 3-CNF \( \phi \) on \( n \) bits, \( \exists C_n \) s.t. \( \phi(C_n(\phi)) = 1 \).

We want to show \( \Sigma_2 = \text{PH} \). It suffices to show \( \Pi_2 \subseteq \Sigma_2 \). Let \( L \) be a complete language for \( \Pi_2 \). We choose take \( L = \Pi_2 \text{SAT} \text{def} \{ \phi \text{ is a 3-CNF : } \forall y \exists z : \phi(y,z) \} \).

The first step is to transform a circuit which merely checks if a circuit is satisfiable, to a circuit that "proves" this by supplying a satisfying assignment.

Claim 10 (Self-reducibility) If \( \forall n \in \mathbb{N} \) there exists a circuit \( C_n \) s.t. \( \phi \in 3\text{-SAT} \iff C_n(\phi) = 1 \) then there exists a circuit \( C'_n \) s.t. \( \phi \in 3\text{-SAT} \iff \phi(C'_n(\phi)) = 1 \) and \( \text{SIZE}(C'_n) \leq \text{SIZE}(C_n)^O(1) \).

Proof Assume circuit \( C \) s.t. \( C(\phi) = 1 \iff \phi \) is satisfiable. If \( \phi \) is satisfiable, discover a satisfying assignment bit-by-bit:

\[
a_1 = \begin{cases} 
0 & C(\phi(0,x_2,\cdots,x_n)) = 1 \\
1 & \text{Otherwise}
\end{cases}
\]

\[
a_2 = \begin{cases} 
0 & C(\phi(a_1,0,x_3,\cdots,x_n)) = 1 \\
1 & \text{Otherwise}
\end{cases}
\]

In general,

\[
a_i = \begin{cases} 
0 & C(\phi(a_1,\cdots,a_{i-1},0,x_{i+1},\cdots,x_n)) = 1 \\
1 & \text{Otherwise}
\end{cases}
\]

At the end, \( \phi(a_1,\cdots,a_n) = 1 \). The construction of the multi-output circuit \( C' \) from the single-output circuit \( C \) is illustrated in Figure 2.

\footnote{Recall that \( L \in \Sigma_2 \) means \( x \in L \iff \exists y \forall z : M(y,z) = 1 \) where \( M \in \text{P}; L \in \Pi_2 \) means \( x \in L \iff \forall y \exists z : M(y,z) = 1 \) where \( M \in \text{P} \). Once we have \( \Pi_2 \subseteq \Sigma_2 \), namely \( \forall \exists \cdots = \exists \forall \cdots \), we can simplify the QBF in the language definition by switching the positions of \( \exists \) and \( \forall \). For example, \( \Pi_4 \exists \forall \exists M(\cdot) = \exists \forall \exists M'(\cdot) = \exists \forall \exists M''(\cdot) = \exists \forall \exists M'''(\cdot) = \exists \forall M'\cdots(\cdot) \in \Sigma_2 \).}
This property is called \textit{self-reducibility}.  \\

\textbf{Algorithm:} Input 3-CNF $\phi$ on $n$ bits, check if $\forall y \exists z \phi(y,z) \equiv \text{iff } C_n(\phi(y,\cdot)) = 1 \iff \text{NP} \subseteq \text{P/poly}$  \\
Hence, we have $\phi \in \Pi_2\text{SAT}$ iff $\exists C' \forall y \phi(y,C'_n(\phi(y,\cdot))) = 1$ \text{in } \Sigma_2$

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.png}
\caption{Self-reducibility: construct $C'$ using $C$ as a building block}
\end{figure}

Next we prove Theorem 9. The intuitive idea is: \textit{“EXP} $\subseteq$ \textit{P/poly}” means any exponential-time computation can be done by a specific poly-size circuit $C_n$ where $n$ is the input size. Then, our goal is to prove that this computation also fits in $\Sigma_2 = \{x: \exists y : |y| \leq |x|^c \forall z: |z| \leq |x|^d \ M(x,y,z) = 1\}$ where $M \in \text{P}$. Recall Cook-Levin Theorem’s proof. An exponential-size computation is hard to check from the outset. But we can make use of the \textit{“for all”} quantifier in $\Sigma_2$. Quantifier $\forall$ comes in handy to check all local places in the exponential-size computation. Meanwhile, the quantifier $\exists$ helps us by “guessing” the circuit $C_n$. $C_n$ is guaranteed to exist when $L \in \text{EXP} \subseteq \text{P/poly}$.

\begin{proof} of Theorem 9: Fix a language $L \in \text{EXP}$. We need to show $L \in \Sigma_2$. Suppose $L$ is computable by a 1-tape TM $M \in \text{TIME}(2^{n^c})$. W.l.o.g., $M$’s alphabet $\Sigma = \{0,1\}$. Let

$\quad L_M \overset{\text{def}}{=} \{(x,i,j) : \text{running } M \text{ on input } x \in L \text{ in time step } i \text{ has a configuration whose } j\text{-th bit is } 1\}$

where in the tuple $(x,i,j)$, $x \in \{0,1\}^n$, $i \leq 2^{n^c}$ denotes $M$’s steps, and $j \leq 2^{n^c}$ is the index of $M$’s $i$-th configuration. Note that we can describe $(x,i,j)$ using poly$(n)$ bits. Clearly, $L_M \in \text{EXP} \subset \text{P/poly}$. So, there is a poly-size circuit $C$ s.t.

$\quad (x,i,j) \in L_M \text{ if } C(x,i,j) = 1$


\begin{itemize}
  \item $M(x)$ at time $N = 2^{n^c}$ entered the accepting state. Namely, $M(x) = 1$.
  \item $x \in L$ if $\exists C \text{ s.t. } C(x,i,j) = 1 \forall i,j \in [2^{n^c}]$.
\end{itemize}

as shown in Figure 3. Note that $x$, $i$, and $j$ can be represented in poly-size.

However, how can we "trust" the circuit $C$? The answer is that, similar to the Cook-Levin theorem, computation is local and can be verified by a collection of local tests (e.g. a CNF). In our case the CNF size is exponential, however we can specify in a uniform way the various tests. These amount basically to
verifying the correct transition between any two time steps. The verifier of the computation simply is "AND of \(2^n\) clauses of constant size". \(\forall i,j \in [2^n]\),

\[
\text{VERIFY} \left( \begin{array}{c}
C(x,i,j), \\
\{C(y,i,\cdot): \text{head in location } j - 1, j, j + 1 \text{ content in location } j\}, \\
\{C(x,i,\cdot): \text{state in step } i\}, \\
\{C(x,i-1,\cdot): \text{state in step } i-1\}
\end{array} \right)
\]

Finally, we renders "\(x \in L \text{ iff } M(x) = 1\)" where \(M \in \text{EXP}\) to

\[
x \in L \text{ iff } \exists C \forall i,j \text{ VERIFY}(C,x,i,j)
\]

VERIFY \(\in \text{P}\) because \(C\) is of poly-size. Therefore, \(L \in \Sigma_2\). This completes the proof. ■

The Karp-Lipton Theorem’s proof is by guessing the assignment. The Meyer’s Theorem’s proof is by verifying the computation in a succinct way.

4 Circuit Lower Bounds

Definition 11 (Size of a circuit) The size of a circuit \(C\) is the number of edges of the circuit \(C\). □

Theorem 12 Most functions on \(n\) bits require size \(\Omega(2^n)\). □

Proof By a counting argument. How many circuits of size \(S\) are there? we need to specify \(S\) wires (which requires \(O(S \log S)\) bits) and the gate computed by each gate (which takes an additional \(O(S)\) bits). Hence the number of functions computed in \(\text{SIZE}(S)\) is at most \(S^{O(S)}\). On the other hand, the number of boolean functions on \(n\) bits is \(2^{2^n}\). Setting \(S = O(\frac{s}{\log n})\), we get that most functions require size \(> S\). ■

Corollary 13 \(\text{SIZE}(\frac{S}{\log S}) \subsetneq \text{SIZE}(S \cdot \log S)\). □

Proof Let \(S = 2^k\), \(k < n\). Most functions on \(k\) bits have size \(> \frac{2^k}{e}\). All functions on \(k\) bits have CNF size \(2^k \cdot k\). ■
References