1 Immerman-Szelepcsényi\footnote{Slovak pronunciation: [ˈʃarɛtʃ ˈzelepʃɛɲi]} Theorem

Theorem 1 (Immerman-Szelepcsényi Theorem [3, 1, 4])

\[\text{co-NL} = \text{NL}\]

Namely, given \(G\), vertices \(s, t\) in \(G\). There is a proof that there is no path \(s \rightarrow t\) that can be verified in logspace.

**Proof** Since \(\text{PATH}\) is co-NL-complete, it suffices to prove that \(\text{PATH} \in \text{NL}\). We’ll use the the read-once certificate definition of \(\text{NSPACE}(i)\). Let \(C_i \stackrel{\text{def}}{=} \{\text{vertices reachable from } s \text{ with } \leq i \text{ steps}\}\). Clearly, \(C_0 = \{s\}\). To prove “vertex \(x \in V\) can be reached in at most \(i\) steps starting from \(s\)” \((x \in C_i)\), it’s easy by providing the path as a certificate: \(s, v_1, \ldots, v_\ell, x\) where \(\ell < i\). The verifier simply checks the vertex chains: \((s, v_1), (v_1, v_2), \ldots, (v_\ell, x) \in G\).

**Intuitive proof idea:** We build a “long” (still poly-size) certificate by concatenating a series of \(n + 1\) sub-certificates, each of which depends on its predecessor sub-certificate excepting for the first one:

\[
\text{Certificate of } |C_0| = 1 \parallel \text{Certificate of } |C_1| = \ell_1 \text{ given } \ell_0 \parallel \cdots \parallel \text{Certificate of } |C_{n-1}| = \ell_{n-1} \text{ given } \ell_{n-2} \parallel \text{Certificate of } t \notin C_{n-1} \text{ given } \ell_{n-1}
\]

where “\(\parallel\)” means concatenation. The idea composes of 2 steps:

- **Step \#1:** Given \(|C_0|\) and a vertex \(x\), prove \(x \notin C_i\).
- **Step \#2:** Given \(|C_{i-1}| = \ell_{i-1}\), prove \(|C_i| = \ell_i\).

Let \(M \in \text{L}\) be the verifier. Although the length of the proof is polynomial, \(M\) never remembers the proof in memory. Lets build the chain of certificates step-by-step. Notice that sub-certificates breaks down to 2 types: the first \(n\) ones and the last one. We use the term “certificate” and “proof” interchangeably.

We will define three types of certificates to make the explanation as smooth as possible; we note that Type-1 certificate can be eliminated and replaced by a type-3 certificate in the overall certificate structure.

**Type-1 certificate construction:** Consider the last sub-certificate. How to construct the certificate that \(x \notin C_i\) given \(\ell_i\)? Say \(C_i = \{y_1, \ldots, y_\ell\}\) are all different from \(x \in V\).

- **Proposal \#1:** \(y_1, \ldots, y_\ell\) \(\parallel\) \(\text{proof } y_1 \in C_i \parallel \cdots \parallel \text{proof } y_\ell \in C_i\).

  \begin{itemize}
  \item \(\text{proof } y_\ell \in C_i\) is simple. It simply gives the path \(s \rightarrow y_\ell\). The verifier \(M \in \text{L}\) checks the path.
  \item However, the “prover” can cheat by offering proofs of \(y' \notin C_i\) because \(M\) cannot remember all vertices shown at the beginning (they can be more than \(O(\log n)\) space). A solution is to “interleave” the vertex being checked and its proof so that its reachability can be checked immediately. This leads to Proposal \#2.
  \end{itemize}

- **Proposal \#2:** \(y_1 \parallel \text{proof } y_1 \in C_i \parallel y_2 \parallel \text{proof } y_2 \in C_i \parallel \cdots \parallel y_\ell \parallel \text{proof } y_\ell \in C_i\)

  The prover can still cheat by giving duplicate \(y_\ell \in C_i\). A simple fix is to require vertices given in ascending order (vertices are numbered).

- **Proposal \#3:** do Proposal \#2 with \(y_1 < y_2 < \cdots < y_\ell\).
Hence, on certificate that $x \notin C_i$ given $\ell_i$, $M$ checks:
- The current vertex being checked, say $y_j$ where $j \in [\ell_i]$. Space: $\log n$.
- The current vertex on the path $s \sim y_j$. Space: $\log n$.
- Make sure $y_1 < y_2 < \cdots < y_{\ell_i}$. No extra space needed.
- The given $\ell_i$. Space: $\log n$.
- A counter remembering the number of vertices already checked. Make sure its final value is equal to the given $\ell_i$. Space: $\log n$.

$M$ needs another $\log n$ space to store $x$. So in total, $M$ uses $5 \log n$ space when it verifies certificate that $x \notin C_i$ given $\ell_i$.

**Type-2 certificate construction:** Then, consider the first $n$ sub-certificates. How to construct the certificate that $|C_i| = \ell_i$ given $|C_{i-1}| = \ell_{i-1}$?

It’s easy to convince $M$ that $|C_i| \geq \ell_i$.

$$y_1 \parallel \text{proof } y_1 \in C_i \parallel \ldots \parallel \text{proof } y_{\ell_i} \in C_i$$

with $y_1 < y_2 < \cdots < y_{\ell_i}$. $M$ verifies this in a way similar to verifying the previous certificate.

Meanwhile, to convince $M$ that $|C_i| \leq \ell_i$, we use the hint $|C_{i-1}| = \ell_{i-1}$. Instead, we show $|C_i| \leq \ell_i$’s equivalent form $\{|1, \ldots, n\} \setminus C_i \geq n - \ell_i$.

$$z_1 \parallel \text{proof } z_1 \notin C_i \text{ given } \ell_{i-1} \parallel \ldots \parallel \text{proof } z_{n-\ell_i} \notin C_i \text{ given } \ell_{i-1}$$

where $z_1 < z_2 < \cdots < z_{n-\ell_i}$. The “no-path” proof is detailed below.

**Type-3 certificate construction:** How to construct the proof $z \notin C_i$ given $\ell_{i-1}$? Notice it’s different from Type-1 certificate (“given $\ell_{i-1}$” instead of “given $\ell_i$”).

$$y_1 \parallel \text{proof } y_1 \in C_{i-1} \parallel \ldots \parallel \text{proof } y_{\ell_{i-1}} \in C_{i-1}$$

where $y_1 < y_2 < \cdots < y_{\ell_{i-1}}$. Hence, on proof $z \notin C_i$ given $\ell_{i-1}$, $M$ checks:
- The current vertex being checked, say $y_j$ where $j \in [\ell_{i-1}]$. Space: $\log n$.
- The current vertex on the path $s \sim y_j$. Space: $\log n$.
- **Extra: checks** $y_j \neq z$ and $(y_j, z) \notin G$. No extra space needed.
- Make sure $y_1 < y_2 < \cdots < y_{\ell_{i-1}}$. No extra space needed.
- The given $\ell_{i-1}$. Space: $\log n$.
- A counter remembering the number of vertices already checked. Make sure its final value is equal to the given $\ell_{i-1}$. Space: $\log n$.

$M$ needs another $\log n$ space to store $z$. So in total, $M$ uses $5 \log n$ space when it verifies proof $z \notin C_i$ given $\ell_{i-1}$.

Obsevant readers may wonder how $\ell_{i-1}$ is given. The answer is that it’s available after the immediate predecessor Type-2 certificate is verified. Note that the above construction is not recursive. Though long, the chain of sub-certificates can be broken down to some basic building blocks. The behavior of $M$ is also clear and it uses $O(\log n)$ memory. This completes the proof.■

### 2 Boolean Circuits

Why time is harder to analyze than space? Combinatorics seem not suitable to reason about the TM configuration graph. Instead, Boolean circuits seem a better model to capture the internal logical structure. They are closer to the real-world implementation, too. For example, even if SAT requires exponential time, it doesn’t rule out the possibility that we can manufacture some special chip that solves SAT-instances of size, say, $10^7$ in just several seconds. A Boolean circuit contains these elements:
• Inputs: \(x_1, x_2, \cdots, x_n\).
• Output(s): one output \(o\) or multiple outputs \(o_1, o_2, \cdots, o_m\).
• Gates: AND, OR, NOT.

Figure 1 shows a circuit that computes the PARITY function.

![Figure 1: A Boolean circuit computing the parity function](image)

**Definition 2 (Boolean circuit)** Boolean circuit with gates \(G\) (e.g., \(G \in \{\text{AND}, \text{OR}, \text{NOT}\}\)) is a DAG. Inputs are \(x_1, \cdots, x_n\). Each note computes some gate \(g \in G\) of its inputs. Output 1 bit. Computing a function \(f : \{0, 1\}^n \rightarrow \{0, 1\}\).

Multi-output Boolean circuits are similarly defined, except that they outputs more than 1 bits.

**Theorem 3 (CIRCUIT-SAT)** Let

\[
\text{CIRCUIT-SAT} \overset{\text{def}}{=} \{C : C \text{ is a circuit with a satisfactory assignment, namely, } \exists x \in \{0, 1\}^n \text{ s.t. } C(x) = 1.\}
\]

CIRCUIT-SAT is NP-complete.

**Sketch of Proof**

(\textbf{In NP}) Obviously, it can be checked in deterministic poly-time.

(\textbf{NP-hard}) Reduce 3-SAT to it. It’s straightforward to build a circuit from a 3-SAT instance \(\phi\) s.t. the circuit has a satisfying assignment iff \(\phi\) is satisfiable. The reduction can be done in poly-time.

**Definition 4 (SIZE(\cdot))** A language \(L \subset \{0, 1\}^*\) is in \(\text{SIZE}(S(n))\) if \(\forall n \in \mathbb{N}, \exists \text{ circuit } C_n \text{ with } n \text{ inputs,}
\)

• \(|C_n| \leq S(n)\), and

• \(\forall x \in \{0, 1\}^n, x \in L \iff C_n(x) = 1.\)

**Definition 5 (P/poly)** Let \(\text{P/poly} \overset{\text{def}}{=} \bigcup_{c \geq 1} \text{SIZE}(n^c)\). \(\text{P/poly}\) is the class of all languages that are computable by poly-size circuits.

The uniform computation model we saw in previous classes considers a single TM that works for inputs of any length. In contrast, the model of Boolean circuits is nonuniform, meaning for inputs of length \(n\), we have a specialized circuit to compute it. The efficient computation notion in the uniform model is the deterministic computation time bounded by a polynomial in \(n\) (\(P = \bigcup_{c \geq 1} \text{TIME}(n^c)\)). Similarly, the efficient computational notion in the nonuniform model is that the circuit size is bounded by a polynomial in \(n\) (\(P/\text{poly} = \bigcup_{c \geq 1} \text{SIZE}(n^c)\)). A natural next question is: what is the relationship between \(P\) and \(P/\text{poly}\)?

**Theorem 6** \(P \subseteq P/\text{poly}\).
Proof Fix input length \( n \). Assume \( \exists \) TM \( M \), running time \( N = n^c \), computing \( L \). Similar to the proof of the Cook-Levin Theorem, recall \( M \)'s computation shown in Figure 2.

Checking the transition from Row \( i \) to Row \( (i+1) \) requires poly-sized circuits. There are \( N \) transitions. So the whole computation can also be done by a poly-sized circuit. Such a circuit can be evaluated in poly-time (this in fact gives an alternative proof to the Cook-Levin Theorem). So for any \( n \), we have poly-sized circuit \( C_n \) to compute \( M(x) \). Therefore, \( P \subseteq P/poly \).

Sidenotes. Back in the 1980s, people were avid in proving circuit lower-bounds. Initially this seemed very promising: various authors were able to show that small circuits of small depth cannot compute specific functions, such as the parity function; and that small monotone circuits cannot compute clique, for example. People believed that it was promising approach to prove \( NP \neq P \). Unfortunately, this attempt turned out to be unsuccessful. The work on natural proofs [2] which we will discuss later showed that certain techniques are doomed to fail, if we believe cryptography exists. We may talk about the limitations of techniques of diagonalization and Boolean circuits towards attacking the \( P \neq NP \) problem later in the course if time permits.

Theorem 7 \( P/poly \) contains uncomputable languages.

Proof Lets prove a claim first.

Claim 8 A language \( L \) is unary if \( L = \{ 1^n : \text{some } n \in \mathbb{N} \} \). A unary language \( L \) is in \( P/poly \).

Sketch of Proof For any \( n \in \mathbb{N} \), let \( C_n \) be

\[
C_n = \begin{cases} 
\text{AND} & 1^n \in L \\
0 & 1^n \notin L 
\end{cases}
\]

We know that Halt is uncomputable. How to encode Halt in a unary language? Consider Halt \(_c \) (TMs that halts on empty input), which can be easily proved to be uncomputable by reducing Halt to it. Let \( L = \{ 1^n : n = \langle M \rangle; \ M \text{ halts on the empty input} \} \).

Definition 9 (\( P \)-uniform) Let \( \{ C_n \}_{n \in \mathbb{N}} \) be a family of circuits where \( C_n : \{0,1\}^n \rightarrow \{0,1\} \). \( \{ C_n \}_{n \in \mathbb{N}} \) is \( P \)-uniform if \( \exists \) TM \( M \) running in poly-time, s.t. \( M(1^n) = C_n \).
Theorem 10  A language $L$ is computable by a $\mathsf{P}$-uniform family of circuits iff $L \in \mathsf{P}$. □

Proof

$(\Rightarrow)$ $L$ is computable by $\{C_n\}_{n \in \mathbb{N}}, \ C_n = M(1^n)$. Then, $x \in L \iff$ Simulate $(M(|x|), x) = 1$.

$(\Leftarrow)$ By the proof of $\mathsf{P} \subset \mathsf{P/poly}$, circuits are uniform. □

References


