P $\neq$ NP

Sheldon S. Nicholl

December 30, 2025

Abstract

Not for publication! Still in draft form; probably contains errors.

In this paper, finite state automata and P-Complete problems are used to study the P versus NP problem. First, all the states at a particular depth of the minimal automaton for a restricted Clique problem are shown to have an in-degree of 1. Next, any automaton for any P-Complete language for strings of the same length is shown to have at least one state at that same level with in-degree greater than 1. So there can be no such circuit and P does not equal NP.

1 Definitions

An *alphabet* is a finite set of characters. A *string* is a finite sequence of characters drawn from an alphabet. A *language* is a set of strings.

A *bit string* is an element of the language $\{0,1\}^*$. If a string $s = vw$, then v is a *prefix* of s and w is a *suffix* of s. The ith character c of string s is accessed with the following notation: $c = s[i]$.

For a language L, define $L_n \subseteq L$ to consist of strings in L of length n:

$$L_n = \{s \mid s \in L \text{ and } |s| = n\} \tag{1}$$

Following Sipser [2], a deterministic finite automaton (DFA) is a tuple

$$A = (\Sigma, S, F, \delta, q_0) \tag{2}$$

where Σ is a finite alphabet, S is a finite set of states, $F \subseteq S$ is the set of final states, $\delta : S \times \Sigma \to S$ is a mapping called the transition function, and $q_0 \in S$ is the initial state.

If $a \in \Sigma$ and the state $s' = \delta(s, a)$ exists, then the pair (s, s') can be viewed as an edge in a graph. So the automaton defines a directed graph with vertices S and edges based on δ. The usual graph concepts like a path, shortest path, and depth then come in their usual meanings. The *depth* of a state $s \in S$ is the length of the shortest path from the start state q_0 to s and will be denoted by $depth(s)$. A *level* is the set of all states in S at a given depth.

If two states q_1 and q_2 are on an accepting path with $depth(q_2) \geq 1$ and a single transition from q_1 to q_2, then q_1 is an *immediate predecessor* of q_2 if $depth(q_1) + 1 = depth(q_2)$.

A Boolean circuit and a finite-state automaton will be called *equivalent* if they decide the same language of bit strings.

2 Automata to Decide Specific k-Clique Problems

Define $\Gamma(n,k)$ to be the set of graphs of size n that have a clique of size k:

$$\Gamma(n,k) = \{g \mid g \text{ contains a clique of size } k, |g| = n\} \tag{3}$$

Define $\Gamma(n)$ to be the set of graphs of size n that have a clique of size $\lfloor \frac{n}{2} \rfloor - 1$:

$$\Gamma(n) = \Gamma(n, \lfloor \frac{n}{2} \rfloor - 1) \tag{4}$$

The set $\Gamma(n)$ is large:

$$|\Gamma(n)| = \Omega(2^n) \tag{5}$$

$\Gamma(n)$ is basically the Half-Clique problem which is shown here to be NP-complete.

2.1 Automaton construction

A finite state automaton can be created from a finite language via the following standard procedure. Since the language is finite, it can be written $\{w_1, w_2, w_3, ..., w_m\}$ where $w_i \in \Sigma^*$. This set can then be turned into the following regular expression: $E = w_1 \cup w_2 \cup w_3 \cup ... \cup w_m$. E is then converted to a non-deterministic finite automaton N. Then N is converted into a deterministic finite automaton D. Finally, D is minimized to yield the minimal automaton [2].

Lemma 2.1. *No deterministic finite-state automaton for a finite language has a cycle along any accepting path.*

Proof. If it had such a cycle, it would accept an infinite language. □

Lemma 2.2. *Let L be a finite language where every string $s \in L$ is of the same length $|s| = n$. Let M be the minimal DFA for L, and let q_1 and q_2 be two states in M such that there is a transition from q_1 to q_2 on an accepting path. Then q_1 is always an immediate predecessor of q_2.*

Proof. Suppose not. Let $depth(q_1) = d$ where $d < n$, and let $depth(q_2) = d'$ with $d' \leq d$. So there is an accepting path of length $d+1+(n-d')$, where d is the depth of q_1, 1 is the length of the transition from q_1 to q_2, and $(n-d')$ is the length of the path from q_2 to an accepting state. This means there is a string $s' \in L$ where $|s'| > n$ since $d+1+(n-d') > n$. But this contradicts the given assumption that $|s| = n$. The proof for "leaps" from q_1 to q_2 where $depth(q_2) - depth(q_1) > 1$ is similar. This lemma can also be proven from the Myhill-Nerode theorem. So the lemma follows. □

A finite-state automaton to decide a specific Clique problem $\Gamma(n, k)$ will now be constructed *for given n and k*. For example, a finite-state automaton can be constructed to determine whether a graph with 10 nodes has a clique of size 4. There is no claim here that a single finite-state automaton can be constructed to decide the Clique problem for *all* n and k.

To perform the construction of such an automaton, an entire graph $g \in \Gamma(n, k)$ will be encoded as a string. An adjacency matrix representation will be used to encode g over the alphabet $\{0, 1\}$, i.e., a bit string.

2.2 Encoding a graph

The encoding of graphs is described below in some detail. An adjacency matrix will be used to encode a graph $g \in \Gamma(n, k)$. The graph in Figure 1 is only intended to illustrate the encoding process.

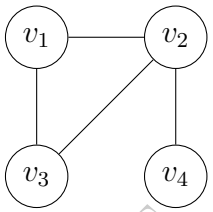

Figure 1: Example Graph

Since the graph is undirected, the adjacency matrix is symmetric, making the top and bottom triangular matrices redundant, so only the upper triangular matrix will be encoded, which looks like Figure 2.

Row-major order will be used to flatten the matrix into a linear form. Consider a row r_i. Each digit in r_i will be treated as a character and all those characters will be concatenated together to form a string s_i. These are shown in Equations 6 through 9.

$$s_1 = 110 \tag{6}$$

$$s_2 = 11 \tag{7}$$

	1	1	0
		1	1
			0

Figure 2: Adjacency Matrix

$$s_3 = 0 \tag{8}$$

$$s_4 = \epsilon \tag{9}$$

Finally all the s_i are concatenated to form the encoding σ:

$$\sigma = s_1 \cdot s_2 \cdot s_3 \cdot s_4 = 110 \cdot 11 \cdot 0 \cdot \epsilon = 110110 \tag{10}$$

Now define a function $e : G \to \Sigma^*$ which maps graphs into their encodings as described above, so for example, $e(g) = \sigma$.

Define $\Theta_{n,k}$ to be the encodings of those graphs $\Gamma(n, k)$:

$$\Theta_{n,k} = \{\sigma \mid g \in \Gamma(n, k) \text{ and } e(g) = \sigma\}$$

2.3 Automata to decide an encoding

There is an automaton to decide $\Theta_{n,k}$. Since $\Theta_{n,k}$ is a finite set of finite strings, it follows that there exists a deterministic finite-state automaton $M_{n,k} = (\Sigma, S, q_0, \delta, F)$ to decide it: $M_{n,k}$ accepts σ iff $g \in \Gamma(n, k)$ and $e(g) = \sigma$. In the running example, the automaton looks like Figure 3.

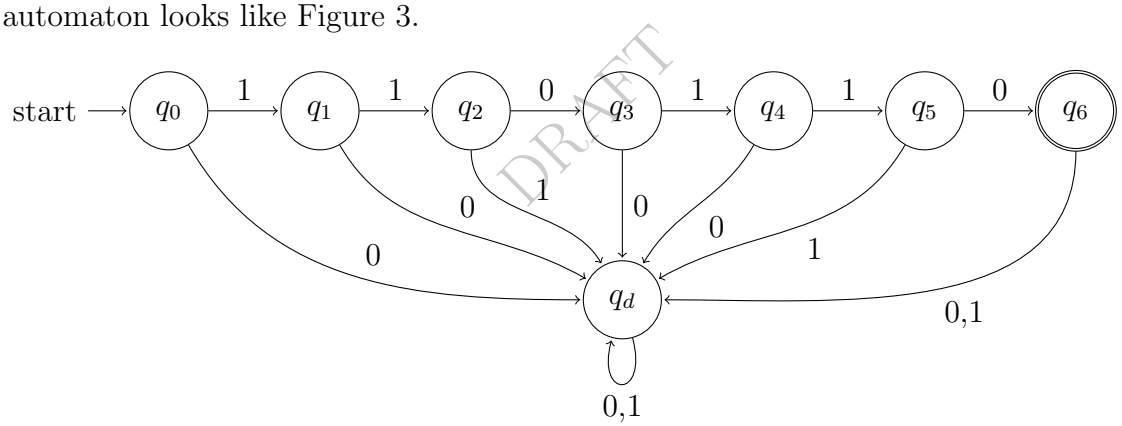

Figure 3: Automaton to accept 110110

2.4 Recoverability Lemma

Lemma 2.3. *If $n \geq 10$ and $k < \lfloor \frac{n}{2} \rfloor$, all the states in the minimal automaton for $\Gamma(n, k)$ at level $k - 1$ in $M_{n,k}$ have in-degree 1, and so do all their ancestor states.*

Proof. Consider a graph $g \in \Gamma(n, k)$, where $g = (V, E)$. Let the vertices in V be numbered, so V can be written $V = \{v_1, v_2, v_3, \ldots, v_n\}$. Suppose that there exists a single clique $C \subseteq V$ where $|C| = k$, $k < \lfloor \frac{n}{2} \rfloor$, and one of the edges of C occurs among the first k vertices of V. Let $m = k - 1$. Let $e(g) = s$ be the string encoding of g, and let s_m be the prefix of s such that $|s_m| = m$ and $s = s_m \cdot s_b$. So s_m encodes the $k - 1 = m$

$$
\begin{pmatrix}
 & \overbrace{}^{m} & & & & & & & \\
- & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & - & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & - & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & - & 0 & 0 & 0 & 0 & 0 & 0 \\
 & & & & - & 1 & 0 & 0 & 0 & 0 \\
 & & & & & - & 0 & 0 & 0 & 0 \\
 & & & & & & - & 0 & 0 & 0 \\
 & & & & & & & - & 0 & 0 \\
 & & & & & & & & - & 1 \\
 & & & & & & & & & -
\end{pmatrix}
$$

Figure 4: Upper triangular adjacency matrix with first m elements in box

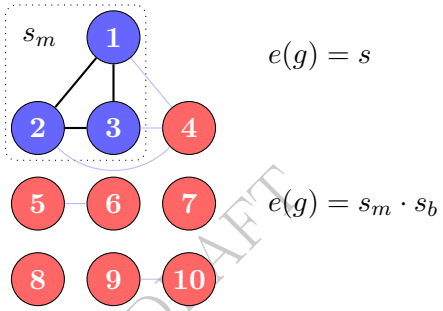

Figure 5: Graph g with a 3-vertex clique

edges $(1,2),(1,3),(1,4),\ldots,(1,k)$, and at least one of those edges is in clique C. The other substring s_b encodes all the other edges, including all the edges between nodes $\{v_2, v_3, \ldots, v_n\}$. While there may indeed be a k-clique encoded in s_b, it is also possible that there is no such k-clique encoded in s_b; this suffices to create the Myhill-Nerode distinguisher below. See Figures 4 and 5 for a visualization of this, where $n = 10$, $\lfloor \frac{n}{2} \rfloor = 5$, $k = 4$, and $m = 3$, $V = \{v_1, v_2, v_3, \ldots, v_{10}\}$, and $C = \{v_1, v_2, v_3, v_4\}$.

Now consider a graph $g' = (V, E')$ which is equal to g except one edge is missing as will be described. See Figure 6 for a visualization of this with the edge between nodes 1 and 2 removed. Let $e(g') = s'$ be the string encoding of g', and let s'_m be the prefix of s' such that $|s'_m| = m$ and $s' = s'_m \cdot s'_b$. Now choose z where $1 \leq z \leq m$, $s[z] = 1$ and let $s'[z] = 0$ such that the clique C is gone from g' with all the other bits of s and s' equal.

So the clique is gone in g', and by the difference between $s[z]$ and $s'[z]$ we have that $s \neq s'$. But nothing else in the encodings has changed, so $s_b = s'_b$, which implies $s' = s'_m \cdot s_b$.

This is the setup for the Myhill-Nerode argument that follows. Since there is a clique in g, there is the membership $s_m \cdot s_b \in \Theta_{n,k}$. But $s'_m \cdot s_b \notin \Theta_{n,k}$ since there is no clique in g'. These two membership relations show that s_b is a Myhill-Nerode distinguishing extension for s_m and s'_m. So s_m and s'_m cannot be in the same Myhill-Nerode equivalence class. Therefore by the Myhill-Nerode theorem, they cannot be in the same state in a

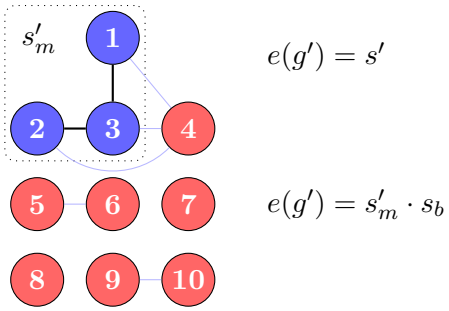

Figure 6: Graph g with 3-vertex clique removed

minimal deterministic finite-state automaton to recognize $\Theta_{n,k}$. So all the states at level $k-1$ are pairwise distinct, and therefore their in-degree must be 1. The ancestor states must also all be distinct since otherwise there would still be pairs of indistinguishable states at level $k-1$.

Finally, Lemmas 2.1 and 2.2 prevent incoming transitions from any state in the automaton other than an immediate predecessor, so the lemma follows.

□

Remark. The order of k is bounded by $\lfloor \frac{n}{2} \rfloor$ to ensure there are enough Myhill-Nerode distinguishers to distinguish all the prefixes.

3 Relation to P-Complete languages

Consider a family $\mathscr{L} = \{L_1, L_2, L_3, \ldots\}$ of languages where the subscripts describe the length of the strings in the languages as defined in Equation 1. Assume that $\cup L_n$ is not a sparse language and that all the L_n are decidable in polynomial time. So by definition, there is a reduction from any language in $\mathscr{L}$ to a P-complete language like the Circuit Value Problem, where $\{C_1, C_2, C_3, \ldots\}$ is a family of Boolean circuits. Each Boolean circuit C_n has only a polynomial number of gates as a function of n: $|C_n| = O(n^k)$ for a constant k. There is a reduction from Circuit Value to Unit Resolution [1], so most of the reasoning here will be based on these two problems.

Discussion. To properly encode an element of a language, there must be enough bits in the encoding to distinguish all the elements of the language. For a language with 2^n elements, n bits are required. A binary encoding contains one bit per binary symbol, so it's n binary digits:

$$\frac{\log(2^n)}{\log(2)} = n$$

An encoding alphabet with 4 symbols requires this many such 4-symbols:

$$\frac{\log(2^n)}{\log(4)} = \frac{n}{2}$$

An encoding alphabet with n^k symbols requires this many such symbols:

$$\frac{\log(2^n)}{\log(n^k)} = \frac{n}{k\log(n)}$$

Lemma 3.1. *Given a set of languages* $\{L_1, L_2, L_3, \ldots, L_n, \ldots\}$ *where each* L_n *consists of* $\Omega(2^n)$ *binary strings, the corresponding set of alphabets* $\{\Sigma_1, \Sigma_2, \Sigma_3, \ldots, \Sigma_n, \ldots\}$ *where* $|\Sigma_n| = O(n^k)$ *are insufficient to encode* L_n *if the encoding* $|e_n| = o(n/log(n))$.

Proof. From the foregoing discussion, the conclusion follows that fewer symbols than required creates an insufficient encoding. In particular, $|e|$ symbols drawn from a language of n^k symbols are insufficient to encode 2^n language elements if

$$|e| < \frac{n}{k\log(n)}$$

This can be generalized to sets of languages. Let e_n be an encoding from Σ_n for a string in L_n. Since e_n is insufficient if the following holds,

$$\lim_{n\to\infty} \frac{|e_n|}{n/(k\log n)} = 0$$

the lemma follows by the definition of little o:

$$|e_n| = o(n/log(n))$$

□

Now consider a set of clauses to decide L_n using unit resolution. The inputs will be encoded with unit clauses. Since the alphabet of L_n is $\{0, 1\}$, a clause for 1 can be positive while a clause for 0 will be negative. A string in the language to be recognized just consists of ones and zeros, which will be encoded into the unit resolution formalism as a particular sequence of unit clauses.

Lemma 3.2. *Given a set of clauses* R_n *which decides a language* L_n *of* $\Omega(2^n)$ *binary strings with* $\Omega(2^{\frac{n}{2}})$ *prefix strings of size* $\frac{n}{2}$ *using unit resolution where* $|R_n| = O(n^k)$ *for fixed k, a set of resolvents* R'_n *cannot encode an arbitrary* $\frac{n}{2}$*-bit binary prefix string from* L_n *if* $|R'_n| = o(n/\log(n))$. *Otherwise at least two of the* $\frac{n}{2}$*-bit strings will receive the same encoding. Since* $\lfloor\frac{n}{2}\rfloor - 1 = O(\frac{n}{2})$*, this result applies to prefix strings of size* $\lfloor\frac{n}{2}\rfloor - 1$ *as well.*

Proof. Since $|L_n| = \Omega(2^n)$, it requires $\Omega(n)$ bits of information to distinguish all the strings in L_n correctly. But since $|R_n| = O(n^k)$, only $O(\log(n))$ bits of information come from each element of R_n. So by Lemma 3.1, as n gets large enough, there will be a situation where multiple bit strings will map to a single sequence of non-terminals since $|v_n| = o(n/\log(n))$. □

Definition 3.1. A **divergent clausal language family** is a family

$$\mathscr{L} = \{L_1, L_2, L_3, \ldots\}$$

of languages over bit strings where (1) $\cup L_n$ is not sparse, and (2) each L_n is decidable under unit resolution by a set of propositional clauses R_n that is polynomially bounded, i.e., $|R_n| = O(n^k)$.

4 Clause Lemma

To explain Lemma 4.1, an example is provided with a tree of all possible unit resolution operations against a starting clause $\{\neg a, \neg b\}$ using the unit propagation strategy as shown in Figure 7. As shown in the figure, only the left branch of the tree leads to the empty clause by the resolution rule. As shown in the first node in the right branch of the tree, the unit propagation strategy dictates that a unit clause will justify the erasure of a larger clause if the larger clause contains a proposition identical to the unit clause.

In Figure 8, the unit clauses are replaced by binary digits, where the depth of the connecting edge is assigned to a specific proposition or its negation. Finally this tree is replaced by the equivalent automaton in Figure 9. Since the empty clause is the only indication of a successful contradiction proof, the empty clause will be considered the only final state in an automaton. Since only the left branch of Figure 7 leads to an empty clause, that will be the only accepting path in the automaton, and the other states all merge into a sink state as shown in Figure 9.

This reasoning is generalized and applied in Lemma 4.1.

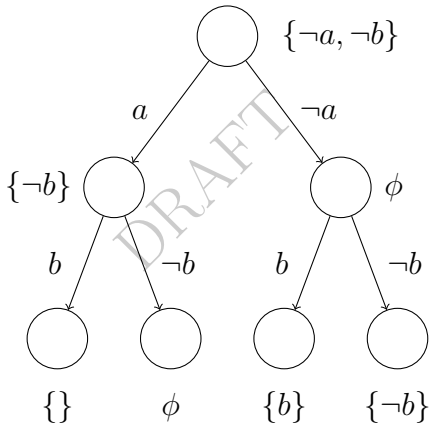

Figure 7: Tree of resolutions

Lemma 4.1. *Let $m = \lfloor \frac{n}{2} \rfloor - 1$. Given a language L_n in a divergent clausal language family and n sufficiently large, the finite-state automaton to decide L_n must always have a state at level m with in-degree greater than 1.*

Proof. Since L_n is finite, it can be recognized by a finite-state automaton, but the construction here will model the process of unit resolution on a finite language. Equation 2 showed the definition of a deterministic finite-state automaton, which will be repeated here: $A = (\Sigma, S, F, \delta, q_0)$. The alphabet is binary since L_n is in a divergent clausal language family, so $\Sigma = \{0, 1\}$. A state is a set of clauses to be defined shortly. If I is the set of input unit clauses and C_n is the set of clauses to decide language membership, then the start state q_0 then is just $I \cup C_n$. The final state is just the empty clause, so the set of final states F is just a set containing the empty clause: $F = \{\{\}\}$. A transition in

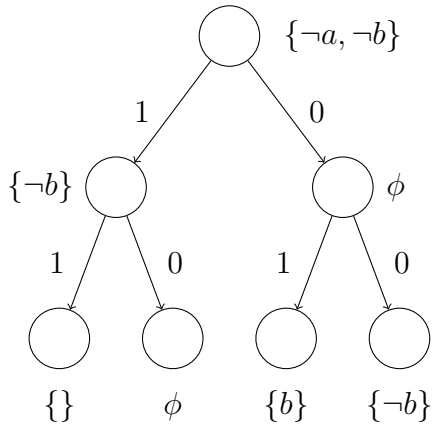

Figure 8: Changing to 0, 1 choices

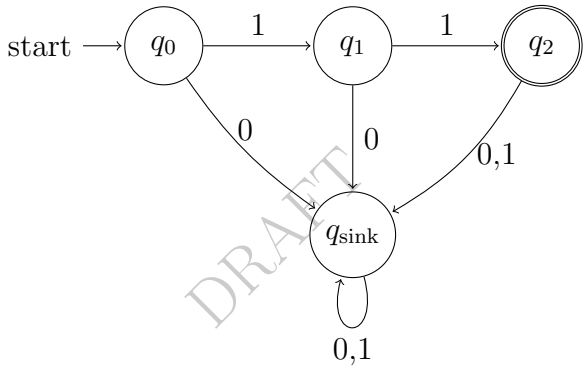

Figure 9: Equivalent Automaton

the transition function δ will model resolution against a unit clause as follows. The start state q_s of the transition will be a set of clauses. The character will be 1 or 0 mapped to a unit clause c. The target state q_t of the transition will be that set of clauses arising from the application of unit propagation to the clauses in q_s with the unit clause c, so the transition is $q_s \xrightarrow{c} q_t$. The transition function consists of all such transitions starting from the start state. Finally, the set of all states S will be the transitive closure of all the possible transitions from the start state.

So by Lemma 3.2 with n sufficiently large, there must be a set of clauses R such that multiple inputs of length m land there. But since R is just a state in the automaton construction of the previous paragraph, this state, which is also at level m, must have in-degree greater than 1.

Now consider the construction of the minimal finite-state automaton from the automaton just given. Suppose two distinct partial inputs i_1 and i_2 yield R and yet somehow land in two different states q_1 and q_2 in the automaton construction above. Suppose c is any possible remaining set of clauses to be read after R is reached. Since q_1 and q_2

both correspond to R, every set of clauses c yields the same outcome (either empty clause [final state] or no empty clause [non-final state]) regardless of whether c comes after q_1 or q_2. This means there are no Myhill-Nerode distinguishers for q_1 and q_2. So by the Myhill-Nerode theorem, q_1 and q_2 must merge into one state in the minimal automaton and the lemma follows.

□

5 Conclusion

Theorem 5.1. $\mathsf{P} \neq \mathsf{NP}$.

Proof. Let $m = \lfloor \frac{n}{2} \rfloor - 1$. Half-Clique can be encoded into a set of languages

$$\mathscr{L} = \{L_1, L_2, L_3, \ldots, L_n, \ldots\}$$

The proof is by reductio ad absurdum. So assume there is a polynomial-time algorithm for Half-Clique.

First, since each L_n in $\mathscr{L}$ is finite, there is a corresponding set of automata where each A_n decides each L_n because of the construction in Section 2.1. Each A_n is constructed directly from L_n with no assumption about polynomial-time computability as shown in Equation 11.

$$A = \{A_1, A_2, A_3, \ldots, A_n, \ldots\} \tag{11}$$

For n sufficiently large, it follows by Lemma 2.3 that there can be no state at level m in any such A_n with in-degree greater than 1.

The reductio ad absurdum argument introduces the assumption that there is a polynomial-time algorithm for Half-Clique. So by Theorem 9.30 of Sipser ([2]), there is a circuit family

$$C = \{C_1, C_2, C_3, \ldots, C_n, \ldots\}$$

to decide the Half-Clique problem as defined in Equation 4 which grows as $|C_n| = O(n^c)$ for some fixed c. Each circuit C_n decides the corresponding language L_n. Under the proven existence of a reduction from Circuit Value to Unit Resolution ([1]), there is a divergent clausal language family R to decide Half-Clique as defined in Definition 3.1:

$$R = \{R_1, R_2, R_3, \ldots, R_n, \ldots\}$$

Lemma 4.1 showed under the polynomial-time assumption that for large enough n, there is a corresponding automaton M_n to decide L_n where there must be a state at level m that has in-degree greater than 1:

$$M = \{M_1, M_2, M_3, \ldots, M_n, \ldots\}$$

This contradiction blocks the existence of any polynomially bounded circuit for $\Gamma(n)$. Since Half-Clique is an NP-Complete problem, this means that $\mathsf{P} \neq \mathsf{NP}$. □

References

[1] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, Oxford, 1995.

[2] Michael Sipser. *Introduction to the Theory of Computation*. Cengage, Boston, third edition, 2013.