On the complexity of intersecting finite state automata

George Karakostas[*]
Princeton University

Richard J. Lipton[†]
Princeton University and Telcordia Applied Research

Anastasios Viglas[‡]
Princeton University

January 18, 2000

Abstract

We consider the problem of testing whether the intersection of a collection of k automata is empty. The straightforward algorithm for solving this problem runs in time σ^k where σ is the size of the automata. In this work we prove that the assumption that there exists a better algorithm solving the FSA intersection emptiness problem implies that non-deterministic time is in subexponential deterministic time and also separates NL from P. Furthermore, under a (more general) non-uniform variant of the assumption mentioned above we can prove that $NL \neq NP$. For this result we will also state a more general lemma that can be used to separate complexity classes starting from upper bounds.

1 Introduction

Separating complexity classes is a major problem in complexity theory. There are only few unconditional results and many open questions. Another major open problem is to give an explicit hard function for the circuit and Turing machine models. In this work, we show a connection between the separation of NL from other complexity classes and the hardness of an explicit problem in P. We will consider the problem of deciding whether the intersection of a collection of k finite state automata is empty. Either this problem requires large circuits or $NL \neq NP$. For the uniform case, either this problem does not have fast algorithms or $NL \neq P$.

Let $F_1, F_2, \ldots, F_k$ be a collection of k finite state automata of size[1] $|F_i| = \sigma$ and consider the problem of checking whether their intersection is empty:

$$\bigcap_{i=1}^{k} L(F_i) \stackrel{?}{\neq} \emptyset$$

where $L(F)$ denotes the language accepted by the automaton F.

The standard algorithm for checking the above intersection, involves constructing the finite state automaton corresponding to the Cartesian product $F = F_1 \times F_2 \times \cdots \times F_k$, and solving the emptiness problem for F: $L(F) \neq \emptyset$. The size of F is $O(\sigma^k)$.

Let $\mathcal{F}$ denote the assumption that there is a better algorithm for checking the intersection emptiness problem for a collection of a fixed number k of automata, namely:

Assumption $\mathcal{F}$: *Let $F_1, F_2, \ldots, F_k$ be k FSA's of size σ. Then there is a deterministic algorithm that can decide whether*

$$\bigcap_{i=1}^{k} L(F_i) \stackrel{?}{\neq} \varnothing$$

[*]Computer Science Department, Princeton University, 35 Olden st., Princeton NJ 08540. email: gk@cs.princeton.edu

[†]Computer Science Department, Princeton University, 35 Olden st., Princeton NJ 08540. email: rjl@cs.princeton.edu.
Supported in part by NSF CCR-9633103.

[‡]Computer Science Department, Princeton University, 35 Olden st., Princeton NJ 08540. email: viglas@cs.princeton.edu.

[1]For simplicity, in this paper the size of an automaton is the number of states. The number of bits required for the description of the automaton is essentially the same times a poly-logarithmic factor, which does not affect our computations.

in time $\sigma^{\frac{k}{f(k)}+d}$, where $f(\cdot)$ is an unbounded function that depends only on k and such that $f(k) = o(k)$; $d > 0$ is a constant.

Based on the assumption $\mathcal{F}$ we can prove the following theorems:

1. There is an algorithm solving knapsack in $O(2^{\epsilon n})$ for all $\epsilon > 0$
2. Integer factorization can also be solved in $O(2^{\epsilon n})$ for all $\epsilon > 0$
3. $NTIME(t) \subseteq TIME(2^{\epsilon t})$ for all $\epsilon > 0$

A slight modification of assumption $\mathcal{F}$ also allows us to separate NL from P.

If we consider a non-uniform version of our assumption i.e. that there exists a "small" circuit that solves the emptiness problem for a collection of FSA's then we can prove that $NL \neq NP$. This result can be proved using a new lemma, that provides a general technique for proving complexity class separations and may be of independent interest. A class separation usually requires a lower bound. The lemma presented in section 5 allows us to prove class separations starting from upper bounds (improved algorithms that imply fast simulations).

2 Knapsack and Factoring

We start by showing the implications of assumption $\mathcal{F}$ for two problems that are considered hard: knapsack and factoring. If $\mathcal{F}$ is true then we can construct better algorithms for solving these problems.

We will consider the following type of Knapsack problem (sub-set sum): Given n numbers $a_1, \ldots, a_n \in Z$ and a number b, check if there exists a boolean vector $x = (x_1, \ldots, x_n)$ such that

$$\sum_{i=1}^{n} a_i x_i = b$$

Assuming that there is an "easy" way of checking the intersection of two automata is empty, we can construct an algorithm solving Knapsack in $2^{n/3}n^{O(1)}$. By choosing a collection of k automata, the resulting algorithm runs in $2^{\epsilon n}n^{O(1)}$ for any $\epsilon > 0$.

Theorem 2.1 *The assumption $\mathcal{F}$ implies that there is an algorithm solving knapsack in $O(2^{\epsilon \cdot n} \cdot n^{O(1)})$ for all $\epsilon > 0$*

Proof Pick two primes[2] p, q of size $n/3$ and build two machines M_p and M_q testing $\sum_{i=1}^{n} a_i x_i \equiv b(mod\ p)$ and $\sum_{i=1}^{n} a_i x_i \equiv b(mod\ q)$. The size (number of states) of these two machines is $|M_p| = O(p \cdot n^{O(1)})$ (same for M_q), where $p < 2^{n/3}$. Consider the intersection $L(M_p) \cap L(M_q)$. If there is a solution to the given knapsack problem then this intersection is nonempty, since $\sum_{i=1}^{n} a_i x_i \equiv b(mod\ p)$ modulo any number p. If, on the other hand, $\sum_{i=1}^{n} a_i x_i \equiv b$ modulo both primes, then $\sum_{i=1}^{n} a_i x_i \equiv b$ or $b - \sum_{i=1}^{n} a_i x_i$ is a multiple of p and q. Define the set S to be the set of exceptions: all binary vectors x for which $b - \sum_{i=1}^{n} a_i x_i \neq 0$ but the sum is zero modulo both primes. The set S is explicit (multiples of p, q) and we can construct a tree-like acceptor. The size of S is less than $2^{n/3}$ since the number of vectors x which make the quantity $b - \sum_{i=1}^{n} a_i x_i$ equal to zero modulo both p and q is:

$$2^n \cdot \frac{1}{2^{n/3}} \cdot \frac{1}{2^{n/3}} = 2^{n/3}$$

Therefore the size of the automaton for S is at most $O(2^{n/3})$. Now Consider the following intersection problem:

$$L(M_p) \cap L(M_q) \cap \bar{S} \stackrel{?}{\neq} \emptyset \qquad (1)$$

If this intersection is non-empty, then there exists a solution to the given knapsack problem. The FSA for S is acyclic (tree-like). We can compute the intersection of such an acceptor with the automaton M_q without computing the cartesian product of the two machines. We can combine S and any FSA M into a new machine with $|S| + |M|$ states that accepts $L(\bar{S}) \cap L(M)$.

In order to get the desired bound ($2^{\epsilon n}$ for any positive ϵ), use a similar construction for k automata: pick k primes $p_1, \ldots, p_k$ of size $\frac{n}{k+1}$ and follow the same ideas described above to construct k automata M_{p_k}

[2] p and q only need to be relatively prime

that will check if $b - \Sigma_{i=1}^{n} \equiv 0(\mod p_k)$. Construct the tree-like acceptor S as before, and consider the emptiness problem for the intersection

$$\bigcap_{i=1}^{k} M_{p_i} \cap \bar{S} \tag{2}$$

The size of the automata M_{p_k} is at most $\sigma = O(2^{\frac{n}{k+1}})$. S has also size σ, and will be combined in an automaton for $M_{p_k} \cap S$ of size $O(\sigma)$. Now we can use the assumption $\mathcal{F}$: for a collection of k automata of size σ the emptiness problem of their intersection can be solved in time $O(\sigma^{\frac{k}{f(k)}+d})$. This will give us the following upper bound:

$$\exp\left(\frac{k}{k+1}\frac{1}{f(k)}n + \frac{d}{k+1}n\right) \tag{3}$$

For large enough k the above expression becomes

$$2^{\frac{1}{f(k)}n} \tag{4}$$

(assuming that $f(k)$ grows slower than $O(k)$.) Since $f(k)$ is unbounded, $\frac{1}{f(k)}$ can become less than any constant $\epsilon > 0$ by choosing k appropriately. ▮

Using the same ideas as above, we can prove that integer factoring of an n-bit number is solvable in $O(2^{\epsilon n})$, for any $\epsilon > 0$, provided that the assumption $\mathcal{F}$ is valid. Finding deterministic algorithms is a major open problem: The best known deterministic algorithm runs in time $2^{\frac{1}{4}n}$. Even with the Extended Riemann Hypothesis this bound only improves to $2^{\frac{1}{5}n}$ (See [Bac90]).

The problem is the following: Given any integer z of size n, find x, y such that $x \cdot y = z$.

Theorem 2.2 *The assumption $\mathcal{F}$ implies that factoring can be solved in time $O(2^{\epsilon n})$ for any $\epsilon > 0$.*

Proof We will show how to build a fixed number of finite state automata to check if $x \cdot y = z$. Exactly as in the case of the knapsack problem, pick two primes p and q of size $n/3$ and consider the corresponding FSA's M_p, M_q, checking whether $x \cdot y \equiv z(mod\ p)$ and $x \cdot y \equiv z(mod\ q)$ respectively. Again we build the set S of those strings x, y for which $x \cdot y \equiv z$ modulo both primes p and q but $x \cdot y \neq z$. The size of S is $|S| \leq 2^{n/3}$, and since S is explicit we can construct a tree-like FSA acceptor. The following emptiness problem solves the factoring problem:

$$L(M_p) \cap L(M_q) \cap \bar{S} \overset{?}{\neq} \emptyset \tag{5}$$

The input of the finite state machines is the string $x\#y$. Since M_p and M_q are finite state automata, we need to know the length of x and y in advance; we need to know where the string x stops and y starts. Since the length of the factors x, y is not known in advance, we will simply check all possible lengths $|x| = n/2, |x| = n/2 - 1, \ldots$. The size of M_p is $p \cdot n^{O(1)}$ (and $M_q = q \cdot n^{O(1)}$. Based on the assumption $\mathcal{F}$, we can check the intersection 5 for emptiness in $2^{n/3} \cdot n^{O(1)}$.

Now consider k primes $p_1, \ldots, p_k$ of size $\frac{n}{k+1}$ each, build the corresponding collection of FSA's $M_{p_1}, \ldots, M_{p_k}$. The size of each automaton is $|M_{p_i}| \equiv \sigma = p_i \cdot n^{O(1)} = 2^{\frac{n}{k+1}} \cdot n^{O(1)}$. The factoring problem can be solved by checking the following intersection:

$$\bigcap_{i=1}^{k} M_{p_i} \cap \bar{S} \tag{6}$$

where S is the set of all numbers x, y such that $x \cdot y \equiv z(\mod p_i)$ for $1 \leq i \leq k$, but $x \cdot y \neq z$. S is a tree acceptor FSA, and can be combined with M_{p_k} into one FSA of size $2^{\frac{n}{k+1}}$ as discussed above.

By our assumption $\mathcal{F}$ the intersection from equation (6) can be solved in time $\sigma^{\frac{k}{f(k)}+d}$. This yields the following upper bound:

$$n^{O(1)} \cdot \exp\left(\frac{k}{k+1}\frac{1}{f(k)}n + \frac{d}{k+1}n\right) \tag{7}$$

In order to factor a given number z proceed as follows: check whether there exist x, y such that $xy = z$ (trying all possible lengths for x and using the automata intersection technique presented above). In order to find the actual number x, compute its bits one by one by solving the following problem: is there a factorization of $z = xy$ where the first bit of x is 1? If we repeat this $O(n/2)$ times, we can find all the bits of the number x. ∎

3 Non-deterministic time t versus deterministic time $2^{\epsilon t}$

In this section we will show how to build a collection of automata to check deterministically the computation of a non-deterministic Turing machine. Under the assumption $\mathcal{F}$ the time required for constructing the automata and checking their intersection for emptiness will be subexponential.

We will use the notion of *block respecting* computation introduced by Hopcroft Paul and Valiant in [HPV77] to prove that deterministic space is strictly more powerful than deterministic time ($DTIME(t) \subseteq DSPACE(t/\log t)$). Block respecting Turing machines are also used in [PPST83] to prove that non-deterministic linear time is more powerful than deterministic linear time (see also [PR81] for a generalization of the results from [HPV77] for RAMs and other machine models).

Definition 3.1 *Let M be a machine running in time $f(n)$, where n is the length of its input x. Let the computation of M be partitioned in $a(n)$* segments, *where each segment consists of $b(n)$ consecutive steps ($a(n) \cdot b(n) = f(n)$). Let also the cells of the tapes of M be partitioned into $a(n)$* blocks *each consisting of $b(n)$ cells on each tape. We will call M* block respecting *if during each segment of its computation, each head visits only one block on each tape.*

The main idea is the following: Let M be a non-deterministic Turing machine running in time t. On input x, convert M to a block respecting machine M_b. A *trace of computation* on input x of a machine M is a string of computation steps. Each step contains the current contents of the working tapes, the position of the heads, the state of M, the input symbol read, the position of the head on the input tape, and the non-deterministic choice of M at this step. Consider the trace of the computation of M_b and construct a collection of FSA's that will check if the computation is correct the following way: each FSA will check the correctness for a number of segments of the computation trace. Note that since M_b is block respecting, for each segment of the computation, the automaton needs to check the contents of only one block on each working tape. Now consider the intersection of all the automata (an automaton accepts its input if it corresponds to a valid computation of M_b on x). If the intersection of the automata is non-empty, then there exists a valid accepting computation for M_b and therefore for M.

In order for an FSA to check a computation segment, it needs to know the contents of the corresponding blocks the last time they were accessed in the computation trace. Since the machine M_b is nondeterministic, we need to consider many possibilities for the position of these previous accesses in the computation trace for M_b. These dependencies can be represented by a graph as in [HPV77]. For the deterministic simulation we need to consider all possible graphs.

Theorem 3.2 *Assumption $\mathcal{F}$ implies $NTIME(t) \subseteq DTIME(2^{\epsilon t})$, for any $\epsilon > 0$.*

Proof Let M be a non-deterministic machine with l tapes, running in time t. Let M_B be the corresponding block respecting machine, with running time $O(t)$. Break the computation of M_B (on input x) in segments of size B each; the number of segments is $O(t/B)$. Consider the directed graph G corresponding to the computation of the block respecting machine as described in [HPV77]: G has one vertex for every time segment (that is t/B vertices) and the edges are defined from the sequence of head positions. Let $v(\Delta)$ denotes the vertex corresponding to time segment Δ then and Δ_i is is the last time segment before Δ during which the i-th head was scanning the same block as during segment Δ. Then the edges of G are $v(\Delta-1) \to v(\Delta)$ and for all $1 \le i \le l$, $v(\Delta_i) \to v(\Delta)$. The number of edges can be at most $O((l+1)\frac{t}{B})$ and therefore the number of bits required to describe the graph is $O\left((l+1)\frac{t}{B}\log\frac{t}{B}\right)$[3].

The general idea for simulating M_B, is to build finite state automata to check the computation that takes

[3]Since l is a constant, from now on it will be incorporated in the big-O notation.

place on each vertex of the graph (each vertex corresponds to a segment of the computation). Since the machine M is non-deterministic, we will need to consider all $2^{O\left(\frac{t}{B}\log\frac{t}{B}\right)}$ possible such graphs.

Now consider the computation of the block respecting machine during time segment Δ: this time segment contains B computation steps. In each step, the machine reads and writes the bits on the head positions in the blocks corresponding to Δ depending on the non-deterministic choice at that step.

In order to check if the computation is correct during one step, we could use an FSA of constant size (the size actually depends only on the number of tapes of the machine). This check can be done by a decision tree of size $2^{O(B)}$.

Let k be the number of FSA's. Then for any $\epsilon > 0$ we can pick $k = 1/\epsilon$ such that each automaton checks $\frac{1}{k}\frac{t}{B}) = \epsilon t/B$ segments or $\frac{\epsilon t}{B}B = \epsilon t$ steps. The size of each FSA is therefore $2^{\epsilon t}$ (decision tree).

Our deterministic algorithm that will simulate M must construct the transition diagrams for these k FSA's. For each transition (arc in the decision tree) we need to simulate M_b for at most 2^B steps. Since $2^{\epsilon t}$ is the total number of transitions, the total time required is $2^{\epsilon t} \cdot 2^B$. The running time for the construction of all the FSA's for all possible graphs is therefore:

$$2^{\epsilon t} 2^B 2^{O(\frac{t}{B}\log\frac{t}{B})} = 2^{O(\epsilon t)} \tag{8}$$

In order to check if there exists an accepting computation for M on input x it suffices to check if the intersection of all FSA's $\cap_{i=1}^{k} F_i$ is non-empty. Under our assumption $\mathcal{F}$, the time needed to intersect k FSA's of size $\sigma = 2^{\epsilon t} = 2^{t/k}$ is:

$$\begin{aligned} \sigma^{\frac{k}{f(k)}+d} &= \left(2^{t/k}\right)^{\frac{k}{f(k)}+d} \\ &= 2^{\frac{1}{f(k)}t + \frac{d}{k}t} \end{aligned} \tag{9}$$

Since $f(k) = o(k)$, the time needed for testing the intersection for emptiness is $2^{O(\frac{1}{f(k)}t)}$. Therefore the total time for our simulation is the time to construct the FSA's plus the time to check the intersection i.e. $2^{O(\epsilon t)} + 2^{O(\frac{1}{f(k)}t)}$. Since $f(k)$ is unbounded we can always pick k appropriately so that the total time is $2^{O(\epsilon t)}$. ∎

4 Uniform assumption: NL vs. P

Till now our assumption $\mathcal{F}$ was that given k FSA's $F_1, F_2, \ldots, F_k$ of the same size σ, there is an algorithm that can check whether their intersection is empty in time $c_k \sigma^{\frac{k}{f(k)}+d}$. We modify slightly this assumption to the following assumption $\mathcal{F}'$:

Assumption $\mathcal{F}'$: *Let $F_1, F_2, \ldots, F_k$ be k FSA's of size σ and G a FSA of size σ'. Then there is a deterministic algorithm that can decide whether*

$$\bigcap_{i=1}^{k} L(F_i) \cap G \overset{?}{\neq} \varnothing$$

in time $\sigma^{\frac{k}{f(k)}+d}\sigma'$, where $f(\cdot)$ is an unbounded function such that $f(k) = o(k)$, and $d > 0$ is a constant.

Notice that the new assumption $\mathcal{F}'$ differs from $\mathcal{F}$ only in the introduction of an extra FSA G which may not have the same size as the rest of the FSA's. The problem can still be solved by the standard method of taking the Cartesian product of the $k+1$ automata and deciding whether its language is empty in time $O(\sigma^k \sigma')$.

Theorem 4.1 *Assumption $\mathcal{F}'$ implies $NL \subseteq DTIME(n^{1+\epsilon})$, for any $\epsilon > 0$.*

Proof Without loss of generality, we will assume that an NL machine has only one working tape.

The main idea is the following: we break the working tape of the machine into blocks. This corresponds into breaking the computation of the machine into segments. We will use one FSA for each tape block that will accept only strings representing 'correct' computations for this particular block. This is done by having the automaton going through its input (claimed to be a valid computation) until it reads a pair of *(head position, machine state)* indicating that the head of the working tape is entering its tape block. Then the FSA simulates the computation of the machine in this block, and checks whether the input represents exactly this computation. Notice that the automaton 'remembers' the contents in its tape block in its own state, in order to do the simulation the next time it encounters its block in its input.

If there is a string that belongs in the languages accepted by *all* the FSA's (i.e. the intersection of their languages is non-empty), then this string will correspond to a computation that is correct for each block. But there is a caveat: our FSA's will be too small to check whether the input of the machine was read correctly throughout the computation. To do this we need another FSA, hence the modification of assumption $\mathcal{F}$.

More specifically, let $L \in NL$ and M_L be the block-respecting NDTM that accepts it in time $O(n^c)$, using at most $O(c \log n)$ working space, for some constant $c > 0$. A *computation* S_x on input x of this machine (with $|S_x| = O(n^c)$) is a string of computation steps. Each step contains information about the position of the head of the working tape, the state of $M_L(x)$, the input symbol read, the nondeterministic choice of $M_L(x)$ at this step and the symbol read/written on the working tape. Note that S_x does not keep the position of the input tape at each step, but this can be found implicitly by simulating the moves of the input head until that point. Obviously each computation step can be described by at most $O(c \log \log n)$ bits (needed to keep track of the working tape head).

We break the working tape W of $M_L(x)$ into k blocks of size B each (k is a parameter to be determined later). Then $k = O(\frac{c \log n}{B})$. For each block B_i, $i = 1 \ldots k$ we construct a FSA F_i that does the following:

1. F_i reads its input until the working head enters B_i

2. It simulates the computation in B_i until head moves to B_{i-1} or B_{i+1}

3. If the end of computation is reached and the computation simulation is correct then accept/reject according to what $M_L(x)$ does, otherwise go to Step 1.

In order to perform Step 2, F_i has to keep in its state the contents of B_i and the current position of the working head, i.e. it needs to 'remember' $O(c2^B \log \log n)$ bits, and since we are going to pick $B = \Omega(\log \log n)$, the size of F_i is $|F_i| = 2^{O(B)}$. In order to compute the transitions of F_i for a single computation step, we need to run $M_L(x)$ starting from all possible configurations of B_i and for all possible nondeterministic choices at this step. Since the later is a constant, the number of transitions is at most $O(\#\text{ of states of } F_i) = 2^{O(B)}$. Hence the time needed to construct all the F_i's is at most $k2^{O(B)}$.

We deferred till now the question of checking whether an (alleged) computation string S_x reads correctly the input tape of M_L. We cannot assign this task to the F_i's since this requires too many bits ($O(c \log n)$) to keep track of. Thus we construct one more FSA G with $O(n)$ states that reads S_x and just simulates the moves on the input tape.

If the intersection $\cap_{i=1}^{k} L(F_i) \cap L(G)$ is non-empty, then there is a string S_x that represents a correct accepting computation of $M_L(x)$ (as checked by the F_i's) with a legal reading of the input tape (as checked by G). Since we assumed that hypothesis $\mathcal{F}'$ holds, the emptiness of this intersection can be decided in deterministic time

$$\begin{aligned} |F_i|^{\frac{k}{f(k)}+d}|G| &= d_1 2^{d_2(\frac{kB}{f(k)}+B)} n \\ &= d_1 2^{\frac{d_2 c \log n}{f(k)} + \frac{d_2 c \log n}{k}} n \end{aligned} \tag{10}$$

for some constants $d_1, d_2 > 0$. By incorporating in (10) the time needed to construct the FSA's, and since $f(k) = o(k)$, the total time needed for the deterministic simulation is at most

$$n^{\frac{d_3 c}{f(k)}+1}$$

for some constant $d_3 > 0$, and thus we can always pick a big enough k so that $\frac{d_3 c}{f(k)} < \epsilon$, for any $\epsilon > 0$ ($f(\cdot)$ is unbounded). ∎

¿From the well known time hierarchy theorem we get the following:

Corollary 4.2 *Assumption $\mathcal{F}'$ implies $NL \neq P$*

5 Non-uniform assumption: NL vs NP

A non-uniform version of assumption $\mathcal{F}'$ implies that $NL \neq NP$. In order to prove that, we will fist prove a general lemma that can be used to separate complexity classes based on upper bounds (which resembles the line of thought in [IW98]).

Lemma 5.1 *Let $\mathcal{C}_1, \mathcal{C}_2$ be two complexity classes such that:*

1. *$\mathcal{C}_1 \subseteq P/poly$*
2. *if $\mathcal{C}_1 = \mathcal{C}_2$ then for any k, there is a language in $\mathcal{C}_2$ that requires circuits of size $>> n^k$ ($\mathcal{C}_2$ has superpolynomial circuits).*
3. *for some fixed k, $\mathcal{C}_1$ has circuits of size $\leq n^k$ with access to an oracle from $\mathcal{C}_2$.*

Then $\mathcal{C}_1 \neq \mathcal{C}_2$

Proof Let $\mathcal{C}_1 = \mathcal{C}_2$. Consider the fixed polynomial size circuit implied by 3. Since $\mathcal{C}_1 = \mathcal{C}_2$ the $\mathcal{C}_2$ oracle has also (fixed) polynomial size, and therefore all $\mathcal{C}_2$ has fixed polynomial size circuits. But this contradicts 2. ∎

The new, non-uniform assumption $\mathcal{F}_\mathcal{C}$ is the following:

Assumption $\mathcal{F}_\mathcal{C}$: *Let $F_1, F_2, \ldots, F_k$ be k FSA's of size σ and G a FSA of size σ'. Then there is a circuit that can decide whether*

$$\bigcap_{i=1}^{k} L(F_i) \cap G \overset{?}{\neq} \varnothing$$

of size $\sigma^{\frac{k}{f(k)}+d}\sigma'$, where $f(\cdot)$ is an unbounded function such that $f(k) = o(k)$, and $d > 0$ is a constant.

Theorem 5.2 *Assumption $\mathcal{F}$ implies that NL can be simulated by fixed polynomial size (size n^α, for some fixed constant α) circuits.*

Proof The proof is essentially the same as for theorem 4.1. Each of the automata F_i, $i = 1 \ldots k$ is of size $2^{O(B)}$ and thus can be described by a circuit of size $2^{O(B)}$. G can be described by a circuit of size $O(n^2)$. Assuming $\mathcal{F}_\mathcal{C}$, there is a size $|F_i|^{\frac{k}{f(k)}+d}|G| = n^{O(\frac{c}{f(k)}+\frac{c}{k})}$ circuit that given the description of automata F_i, $i = 1 \ldots k$, and G from theorem 4.1 decides the emptiness of their intersection. By picking k large enough this size can be made equal to n^α for some constant $\alpha > 0$ independent of c. Hence every language in NL has a circuit of size n^α.

∎

Corollary 5.3 *Assumption $\mathcal{F}_\mathcal{C}$ implies $NL \neq NP$*

Proof

1. NL has polynomial size circuits.
2. If $NL = NP$ then the polynomial time hierarchy collapses to NL, and Kannan's theorem from [Kan81] implies that for any constant β there is a language in Σ_4 and therefore NL that is not computable by circuits of size n^β.
3. By theorem 5.2 NL has fixed polynomial (n^α) size circuits.

Therefore, lemma 5.1 implies that $NL \neq NP$. ∎

Note that in the proof of corollary 5.3 we do not use the full power of lemma 5.1. In fact Kannan's theorem would suffice. It would be interesting to see applications of lemma 5.1 in its full power to separate complexity classes.

6 Conclusions

In this work we presented a number of interesting results based on the hardness of solving the emptiness problem for the intersection of finite state automata. Lemma 5.1 provides a general method for separating complexity classes starting from upper bounds and might enable us to prove other interesting separations.

The methods used to prove our results are very classic in complexity theory. We think that they still have not been exploited in their full power. For example it would be interesting to see more applications of lemma 5.1 for separating other complexity classes.

Acknowledgments

Many thanks to Sanjeev Arora.

References

[Bac90] Eric Bach. Number-theoretic algorithms. In *Annual Review of Computer Science*, volume 4, pages 119–172. Annual Reviews, Inc., 1990.

[HPV77] J. Hopcroft, W. Paul, and L. Valiant. On time versus space. *Journal of the ACM.*, 24(2):332–337, April 1977.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: De-randomization under a uniform assumption. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 734–743, Los Alamitos, CA, November8–11 1998. IEEE Computer Society.

[Kan81] R. Kannan. A circuit-size lower bound. In *22th Annual Symposium on Foundations of Computer Science*, pages 304–309, Los Alamitos, Ca., USA, October 1981. IEEE Computer Society Press.

[PPST83] Wolfgang J. Paul, Nicholas Pippenger, Endre Szemerédi, and William T. Trotter. On determinism versus non-determinism and related problems (preliminary version). In *24th Annual Symposium on Foundations of Computer Science*, pages 429–438, Tucson, Arizona, 7–9 November 1983. IEEE.

[PR81] W. Paul and R. Reischuk. On time versus space II. *Journal of Computer and System Sciences*, 22(3):312–327, June 1981.