The Emptiness Problem for Intersections of Regular Languages*

Klaus-Jörn Lange Peter Rossmanith

Institut für Informatik, Technische Universität München
Arcisstr. 21, D-8000 München 2

Abstract

Given m finite automata, the emptiness of intersection problem is to determine whether there exists a string which is accepted by all m automata. In the following we consider the case, when m is bounded by a function in the input length, i.e., in the size and number of the automata. In this way we get complete problems for nondeterministic space-bounded and timespace-bounded complexity classes. Further on, we get close relations to nondeterministic sublinear time classes and to classes which are defined by bounding the number of nondeterministic steps.

Introduction

There is a general relationship between emptiness problems of formal languages and word problems of complexity classes as noted by Hunt ([10], see also [6]). In the case of finite automata this means that the problem of determining whether a deterministic finite automaton accepts a nonempty language is complete for nondeterministic logarithmic space [11]. While this holds for both nondeterministic and deterministic finite automata, results change if we restrict them to have a singleton input alphabet, i.e., containing one element only. In this case the nonemptiness problem of nondeterministic automata remains complete for nondeterministic logarithmic space, while it is solvable in deterministic logarithmic space for deterministic automata (in fact, this problem is complete for deterministic logarithmic space with respect to NC^1 reductions). These results still hold true, if we generalize these problems by considering the emptiness of the intersection of several finite automata, as long as their number is fixed. However, the situation changes drastically, if we take arbitrary, unbounded numbers of finite automata. The nonemptiness of intersection problem for both deterministic and nondeterministic finite automata is PSPACE-complete (see [13]). If the common input alphabet of the automata is singleton, the problem is NP-complete for both deterministic and nondeterministic automata [8].

In this work the number of automata will be bounded by some function $g(n)$ in the input length n, which is the length of the codings of these automata. Obviously, any reasonable function g must fulfill $1 \leq g \leq n$ for all n, since we cannot encode

*work supported by the Deutsche Forschungsgemeinschaft, La 618 1-1 and SFB 0342 A4 "KLARA"

more than n automata within n symbols. By the above mentioned results the complexity of the nonemptiness of intersection problem for $g(n)$ automata must lie between NSPACE($\log n$) and PSPACE. In fact, we will show in the next section that this problem is NSPACE($g(n) \cdot \log(n)$)-complete, regardless of the determinism or nondeterminism of the underlying automata. Thus for the first time there are now complete problems which are not purely codings of machines and their inputs for classes like NSPACE($\log^k n$) or NSPACE($\log^{\log \log n} n$). In the case of a singleton input alphabet the situation is a bit less uniform. We will see that the emptiness of intersection problem of $g(n)$ *unary* deterministic finite automata is logspace reducible to a set in NTIME($g^2(n)$), which implies that it is solvable by a polynomial time machine which uses only a small amount of working tape and makes only few nondeterministic steps. Thus this problem is very unlikely to be NSPACE($\log n$)-hard, as long as g is strictly sublinear, e.g. polylogarithmic. On the other hand we can reduce in logarithmic space any set in NTIME($\sqrt{g(n)}$) to this problem. In total these two relations show a very close relationship between unary regular intersections and sublinear NTIME-classes.

Preliminaries

We assume the reader to be familiar with some basic notions of complexity theory (as contained in [9], [2], or [18]). In addition we use the following notation: DTISP(f, g) (NTISP(f, g)) is the class of all languages accepted by deterministic (nondeterministic) Turing machines, which are simultaneously time bounded by f and space bounded by g.

Completeness and hardness are always meant with respect to deterministic many-one logspace reducibilities, unless otherwise stated. $L \leq_{log} M$ means that L is reducible to M.

Finally, a function $g(n)$ is said to be *log-constructible*, if a logspace bounded Turing machine can print $g(n)$ in unary coding (i.e.: $a^{g(n)}$) given the unary input a^n. In general, nearly all of polynomially bounded functions commonly used as resource bounds are log-constructible. That is why we will use throughout of this paper only log-constructible functions:

Assumption. In the following we denote by $g(n)$ only functions $1 \leq g(n) \leq n$ that are log-constructible.

Intersection emptiness of regular languages

In this section we investigate the intersection emptiness problem for a bounded number of deterministic or nondeterministic finite automata. In this way we will get complete problems for classes between NSPACE($\log n$) and NSPACE($n^{O(1)}$) = PSPACE.

These problems have finite automata as inputs. Therefore we need an appropriate coding function $\langle \cdot \rangle$, which maps a sequence $A_1, A_2, \ldots, A_m$ of finite automata into a word $\langle A_1, A_2, \ldots, A_m \rangle$ over a fixed alphabet X. We do not go into the details of $\langle \cdot \rangle$, but assume it to fulfill certain standard properties; for instance that the coding of states and symbols is of logarithmic length. We then set

$$\emptyset \neq \bigcap \text{DFA} := \left\{ \langle A_1, A_2, \ldots, A_m \rangle \;\middle|\; \begin{array}{c} A_i \text{ is a deterministic finite automaton,} \\ 1 \leq i \leq m,\ m \geq 1,\ \bigcap_{i=1}^{m} L(A_i) \neq \emptyset \end{array} \right\}.$$

The set $\emptyset \neq \bigcap$NFA is defined analogously with respect to nondeterministic finite automata. Kozen [13] showed the PSPACE-completeness of $\emptyset \neq \bigcap$DFA. Since $\emptyset \neq \bigcap$NFA can be decided within polynomial space, $\emptyset \neq \bigcap$NFA is PSPACE-complete, too. These intractable problems become feasible, if we bound the number of finite automata by a fixed k. Thus we set

$$\emptyset \neq \bigcap^{k} \text{DFA} := \{\langle A_1, A_2, \ldots, A_k\rangle \mid \langle A_1, A_2, \ldots, A_k\rangle \in \emptyset \neq \bigcap \text{DFA}\}$$

and define $\emptyset \neq \bigcap^k$NFA in a corresponding way. By Galil we know that for each k both $\emptyset \neq \bigcap^k$DFA and $\emptyset \neq \bigcap^k$NFA are NSPACE($\log n$)-complete (see [8]).

In the following we will consider these problems for the case that the number m of finite automata is bounded by a function $g(\cdot)$ in the length $n := |\langle A_1, A_2, \ldots, A_m\rangle|$. [1] That is, we now consider

$$\emptyset \neq \bigcap^{g} \text{DFA} := \left\{ \langle A_1, A_2, \ldots, A_m\rangle \;\middle|\; \begin{array}{c} \langle A_1, A_2, \ldots, A_m\rangle \in \emptyset \neq \bigcap \text{DFA and} \\ m \leq g(|\langle A_1, A_2, \ldots, A_m\rangle|) \end{array} \right\}.$$

Again, $\emptyset \neq \bigcap^g$NFA is defined using nondeterministic finite automata.

The first theorem of this section describes the hardness of intersection emptiness of $g(n)$ deterministic automata:

Theorem 1 NSPACE($g(n)\log n$) $\leq_{\log} \emptyset \neq \bigcap^{g}$DFA.

Proof: Let M be some NSPACE($g(n)\log n$) machine. We construct a function f which is logspace computable and maps $w \mapsto \langle A_1, \ldots, A_{g(n)}\rangle$ to a coding of $g(n)$ DFA's $A_1, \ldots, A_{g(n)}$ such that $w \in L_M$ iff $f(w) \in \emptyset \neq \bigcap^g$ DFA. Each word $\#c_1\#c_2\#\ldots\#c_m \in \bigcap_{i=1}^{g(n)} L_{A_i}$ corresponds to an accepting computation of M with input w, where c_j are configurations of M. One individual DFA A_i checks whether $\#c_1\#\ldots\#c_m$ is a valid computation of M by simulating M using its states to remember M's configuration. Since M has $n^{O(g(n))}$ different configurations, this would, however, require $n^{O(g(n))}$ states, which are too many. So DFA A_i will only remember the state of M, the input head position and an $O(\log n)$ long *part* of M's working tape. In [13] each automaton was responsible for only one symbol of the working tape which would lead in our case to $g(n)\log n$ DFA's rather than only $g(n)$. On the other hand, a "window" bigger than $O(\log n)$ is not possible, if the number of states has to stay polynomial.

The correctness of a computation $\#c_1\#\ldots\#c_m$ of M will be tested by A_i only for the part of the working tape which A_i is responsible for. For $g(n)$ DFA's it is possible to cover the *whole* working tape (in an overlapping manner). Those $\#c_1\#\ldots\#c_m$ accepted simultaneously by all DFA's are then exactly M's accepting computation and each DFA A_i has only a polynomial number of states, so $\langle A_1, \ldots, A_{g(n)}\rangle$ will be polynomially bounded in its length and is therefore logspace computable. □

The next theorem shows that $\emptyset \neq \bigcap^g$NFA is contained in NSPACE($g(n)\log n$).

[1] Here $|v|$ denotes the length of a word v.

Theorem 2 $\emptyset \neq \bigcap^{g} \mathrm{NFA} \in \mathrm{NSPACE}(g(n)\log n)$.

Proof: To simulate one individual NFA it is sufficient to use $\log n$ space. For $\langle A_1, \ldots, A_m \rangle$ simply guess a word $w \in \bigcap_{i=1}^{m} A_i$ symbol by symbol and simulate all m NFA's A_1,...,A_m simultaneously step by step on w. □

Please note that though we have $\emptyset \neq \bigcap^{g} \mathrm{NFA} \in \mathrm{NSPACE}(g(n)\log n)$ it is very unlikely that $\emptyset \neq \bigcap^{g} \mathrm{DFA} \in \mathrm{DSPACE}(g(n)\log n)$ holds due to the completeness of $\emptyset \neq \bigcap^{g} \mathrm{DFA}$ for $\mathrm{NSPACE}(g(n)\log n)$. The "reason" why intersection emptiness of DFA's is as hard as for NFA's can be seen as follows: Nondeterministic guesses are needed to guess a word in the intersection *and* for the NFA's built-in nondeterminism. For DFA's, however, the next state has also to be guessed, since the next input symbol has to be guessed. For a singleton input alphabet this is no longer true; and in fact $\emptyset \neq \bigcap^{g} \text{Tally-DFA} \in \mathrm{DSPACE}(g(n)\log n)$. In the next section we will improve this relation. From Theorems 1 and 2 we get the following corollary:

Corollary 3 *Both* $\emptyset \neq \bigcap^{g} \mathrm{NFA}$ *and* $\emptyset \neq \bigcap^{g} \mathrm{DFA}$ *are complete for* $\mathrm{NSPACE}(g(n)\log n)$.

In particular, we present a first "natural" complete problem for $\mathrm{NSPACE}(\log^k n)$:

Corollary 4 $\emptyset \neq \bigcap^{\log^{k-1} n} \mathrm{DFA}$ *is complete for* $\mathrm{NSPACE}(\log^k n)$, *for* $k \geq 1$.

We proceed with the investigation of a refinement of the intersection emptiness problem: The smallest word in the intersection of the languages of $g(n)$ automata can be as large as $n^{g(n)}$, where n is the length of the coding of all $g(n)$ automata. If we are interested in the *existence of short words* in the intersection of NFA's and DFA's rather than in the mere existence, the problem gets much simpler. In the following we will regard words of linear length as "short". Instead of linear length we could bound the length also by any fixed polynomial without changing the results.

By $\emptyset \neq \bigcap^{g} \mathrm{DFA}_{\mathrm{lin}}$ we denote the language of all codings of up to $g(n)$ DFA's with the property that there is a word w in the intersection of their languages which has *linear length*, i.e., $|w| = O(n)$.

$$\emptyset \neq \bigcap^{g} \mathrm{DFA}_{\mathrm{lin}} := \left\{ \langle A_1, \ldots, A_m \rangle \in \emptyset \neq \bigcap^{g} \mathrm{DFA} \;\middle|\; \exists_{w \in \bigcup_{i=1}^{m} L(A_i)} : |w| \leq |\langle A_1, \ldots, A_m \rangle| \right\}$$

Analogously, we define $\emptyset \neq \bigcap^{g} \mathrm{NFA}_{\mathrm{lin}}$. The languages $\emptyset \neq \bigcap^{g} \mathrm{DFA}_{\mathrm{lin}}$ and $\emptyset \neq \bigcap^{g} \mathrm{NFA}_{\mathrm{lin}}$ will turn out to be complete for simultaneously time and space bounded classes between $\mathrm{NTISP}(\mathrm{pol}, \log n) = \mathrm{NSPACE}(\log n)$ and $\mathrm{NTISP}(\mathrm{pol}, \mathrm{pol}) = \mathrm{NP}$. This leads especially to complete problems for all levels of the NSC-hierarchy, i.e., the nondeterministic analogue of the SC-hierarchy (see [5]).

Theorem 5 $\emptyset \neq \bigcap^{g} \mathrm{NFA}_{\mathrm{lin}} \in \mathrm{NTISP}(\mathrm{pol}, g(n)\log n)$.

Proof: In contrast to the proof of Theorem 2 we have to simulate $g(n)$ NFA's on a word of linear length which is possible in time $ng(n)$. □

We can also prove the hardness of $\emptyset \neq \bigcap^{g} \mathrm{DFA}_{\mathrm{lin}}$:

Theorem 6 $\mathrm{NTISP}(\mathrm{pol}, g(n)\log n) \leq_{\log} \emptyset \neq \bigcap^{g} \mathrm{DFA}_{\mathrm{lin}}$.

Proof: Since $\mathrm{NTISP}(\mathrm{pol}, g(n)\log n) \leq_{\log} \mathrm{NTISP}(\sqrt{n}, g(n)\log n)$ holds, we can start with an $\mathrm{NTISP}(\sqrt{n}, g(n)\log n)$ machine M and proceed as in the proof of Theorem 1. □

Intersection of Regular Tally Languages

This section deals with the intersection emptiness problem of $g(n)$ finite automata with a singleton input alphabet. Thus we will consider the problem

$$\emptyset \neq \bigcap \text{Tally-DFA} := \left\{ \langle A_1, A_2, \ldots, A_m \rangle \in \emptyset \neq \bigcap^{g} \text{DFA} \;\middle|\; \begin{array}{c} A_i \text{ is a deterministic finite} \\ \text{automaton with input alphabet} \\ \{a\} \text{ for } 1 \leq i \leq m \end{array} \right\}.$$

Using nondeterministic finite automata we get $\emptyset \neq \bigcap \text{Tally-NFA}$. As a consequence of [17] and [8] both $\emptyset \neq \bigcap \text{Tally-DFA}$ and $\emptyset \neq \bigcap \text{Tally-NFA}$ are NP-complete.

Correspondingly, we define the problem $\emptyset \neq \bigcap^{k} \text{Tally-DFA}$ and $\emptyset \neq \bigcap^{g} \text{Tally-DFA}$ for an integer k and (log-constructible) function g. Using standard methods, also the $\mathrm{DSPACE}(\log n)$-completeness of $\emptyset \neq \bigcap^{k} \text{Tally-DFA}$ can be obtained for each k. The later result is not simply caused by the use of logspace reducibilities[2]. Depending in the suitable choice of the coding $\langle \cdot \rangle$, these completenesses can be obtained with respect to very fine reducibility types like AC^0 or even DLOGTIME reductions.

The complexities of these problems show tight relationships to classes of *bounded nondeterminism*. So, we will consider machines which are simultaneously bounded in time, space, and number of executed nondeterministic steps. By $\mathrm{TINOSP}(f(n), g(n), h(n))$[3] the class of languages recognized by machines which perform at most $f(n)$ steps, $g(n)$ of which may be nondeterministic, and use at most $h(n)$ space.

A well known example of such a class is the β_k-hierarchy of NP-machines making at most $O(\log^k n)$ nondeterministic steps, i.e., $\beta_k = \mathrm{TINOSP}(\mathrm{pol}, log^k, \mathrm{pol})$, which lies between P and NP and has complete problems for each of its level (see [1]). Other examples of bounded nondeterminism can be found in [3, 7, 12].

We start by stating some simple properties of bounded nondeterminism classes:

$$\mathrm{TINOSP}(—, —, f(n)) = \mathrm{NSPACE}(f(n)),$$

$$\mathrm{TINOSP}(—, 0, f(n)) = \mathrm{TINOSP}(—, f(n), f(n)) = \mathrm{DSPACE}(f(n)),$$

$$\mathrm{TINOSP}(\mathrm{pol}, —, f(n)) = \mathrm{NTISP}(\mathrm{pol}, f(n)).$$

If $f(n) \in o(n)$, it is not known, whether $\mathrm{GAP} \in \mathrm{TINOSP}(\mathrm{pol}, f(n), f(n))$ holds. But if either the space or nondeterminism bound is dropped, GAP is clearly contained in the resulting classes.

The following theorem gives us a first upper bound in terms of TINOSP-classes:

[2] Almost every element of $\mathrm{DSPACE}(\log n)$ is $\mathrm{DSPACE}(\log n)$-complete with respect to logspace reductions!

[3] TINOSP stands for TImeNOndeterminismSPace

Theorem 7 $\emptyset \neq \bigcap^{g} \text{Tally-DFA} \in \text{TINOSP}(\text{pol}, g(n)\log n, g(n)\log n)$.

Proof: First, the length of a word in the intersection can be guessed in $g(n)\log n$ nondeterministic steps and stored within $g(n)\log n$ space. Then it has to be verified, whether the word of this length is accepted by all of the Tally-DFA's. This cannot be done by a simple simulation due to the polynomial time bound. However, since the relevant part of a Tally-DFA corresponds to a transition diagram which consists of a fixed number of state changes ending in a cycle, it is possible to check membership by some simple computations modulo m. □

In the following we will compare emptiness of intersection problems with nondeterministic sublinear time classes. To work with sublinear time makes it necessary to augment the usual model of a Turing machine with an *Index Tape*, which enables the machine to access its input tape more quickly. We refer the reader to [16] and [4].[4]

We already pointed out that $\emptyset \neq \bigcap^{g} \text{Tally-DFA} \in \text{DSPACE}(g(n)\log n)$ holds (see last section) and therefore also $\emptyset \neq \bigcap^{g} \text{Tally-DFA} \in \text{ATIME}((g(n))^2 \log^2 n)$. As an alternative to Theorem 7 we show an improvement of this bound[5]:

Theorem 8 $\emptyset \neq \bigcap^{g} \text{Tally-DFA} \leq_{\log} \text{NTIME}((g(n))^2 \log^2 n)$.

Proof: We use a reduction that maps the coding $\langle A_1, \ldots, A_{g(n)} \rangle$ of the Tally-DFA's to a table that contains $\delta^i_{A_j}(q_{j,k})$ for $i = 1, 2, 4, 8, \ldots, 2^{g(n)\log n}$, where δ_{A_j} is the transition function of A_j and $q_{j,k}$ is the kth state of A_j, i.e., $\delta^i_{A_j}(q_{j,k})$ is the state reached from the state $q_{j,k}$ of A_j in exactly i steps, i.e., after reading a tally word of length i. This table is computable within logarithmic space using modulo computations as in Theorem 7.

Given that table, an NTM can guess the length of a word $w = a^\ell \in \bigcap_{j=1}^{g(n)} L_{A_j}$. It can check, whether $w \in L_{A_j}$ by computing $\delta^\ell_{A_j}(q_{j,0}) = \delta^{2^{i_1}}_{A_j} \circ \delta^{2^{i_2}}_{A_j} \circ \cdots \circ \delta^{2^{i_m}}_{A_j}(q_{j,0})$ with the help of the table. Here $m = g(n)\log n$ and the numbers $i_1, \ldots, i_m$ are directly given by the binary representation of ℓ. If we adopt a model with non-erasing index tape this takes $g(n)\log^2 n$ steps which gives for a total of $g(n)$ Tally-DFA's a running time of $(g(n))^2 \log^2 n$. □

Note that only $g(n)\log n$ nondeterministic steps were used, so it was actually proved

$$\emptyset \neq \bigcap^{g} \text{Tally-DFA} \leq_{\log} \text{TINOSP}((g(n))^2 \log^2 n, g(n)\log n, g(n)\log n).$$

Theorem 7 implied that it is possible to solve $\emptyset \neq \bigcap^{g} \text{Tally-DFA}$ by using polynomial time and a small number of nondeterministic steps *at the end* of the computation rather than at the beginning. Nondeterminism at the beginning of a computation seems to be more powerful than at the end as illustrated by

[4] We should note, that we use the *noneraser* version of an index tape, which is not necessarily erased after each use.

[5] Observe that NTIME(g) is not closed under logspace reducibility unless we choose $g = \text{pol}$

1. space bounded classes [14]: $\mathrm{P} = \mathrm{P}^{\mathrm{NSPACE}(\log n)} \subseteq \mathrm{NSPACE}(\log n)^{\mathrm{P}} = \mathrm{NP}$,

2. bounded nondeterminism [1]:
   $\mathrm{P}(\mathrm{NPOLYLOGTIME}) \subseteq \mathrm{NPOLYLOGTIME}(\mathrm{P}) = \bigcup_{\mathrm{i}>0} \beta_{\mathrm{i}}$,

3. blind nondeterminism [15]: $\mathrm{ENL} \subseteq \mathrm{BNL}$.

We were not able to achieve completeness results for the intersection emptiness problem for tally languages as it was the case for the general problem. We can, however, prove the following hardness result:

Theorem 9 $\mathrm{NTIME}(g(n)) \leq_{\log} \emptyset \neq \bigcap^{G(n)}$ Tally-DFA *for* $G(n) = \dfrac{(g(n))^2 \log g(n)}{\log n}$.

Proof: A computation of a $g(n)$ time bounded NTM M consists of $g(n)$ configurations of size $O(g(n))$, which can be seen as a matrix of $g(n) \times g(n)$ symbols. Let $p_1, \ldots, p_{(g(n))^2}$ denote the first $(g(n))^2$ prime numbers. Let the symbols $S_1, \ldots, S_{(g(n))^2}$ in the matrix be numbers from 0 to k. Then a computation can be coded in unary by $a^{p_1^{S_1} \cdot p_2^{S_2} \cdot \ldots \cdot p_m^{S_r}}$, $r = (g(n))^2$. We will show how to construct for some input word w the encoding $\langle A_1, \ldots, A_m \rangle$ of m Tally-DFA's such that $w \in L_M$ iff $\emptyset \neq \bigcap_{i=1}^{m} L_{A_i}$. For this end we cover the $g(n) \times g(n)$ matrix with small squares of size $\ell \times \ell$.

Each Tally-DFA A_i checks for one small square whether the input a^j could correspond to some accepting computation of M. In order to do this it is necessary for A_i to "know" ℓ^2 certain symbols of the whole square. The ith symbol is exactly $S_i = j \bmod p_i$ which can be computed by a Tally-DFA having p_i states. To compute simultaneously ℓ^2 symbols $S_{i_1}, \ldots, S_{i_{\ell^2}}$ an Tally-DFA with $p_{i_1} \cdot p_{i_2} \cdot \ldots \cdot p_{i_{\ell^2}}$ states is sufficient. The ith prime number p_i is of order $O(i \cdot \log i)$, so we get a Tally-DFA with $O(n)$ states, if we choose $p_{i_1} \cdot p_{i_2} \cdot \ldots \cdot p_{i_{\ell^2}} \in O(p^{\ell^2}_{(g(n))^2}) = O((g(n))^{2\ell^2} \cdot \log^{\ell^2} g(n)) = O(n)$. From this we get $\ell^2 \cdot \log g(n) \in O(n)$ and finally $\ell^2 \in O(\log n / \log g(n))$. In order to cover the whole $g(n) \times g(n)$ square with $\ell \times \ell$ squares we need $O(g(n^2)/\ell^2) = O((g(n))^2 \cdot \log g(n)/\log n)$ individual Tally-DFA's. □

Turning to the nondeterministic case, the situation seems to be more difficult. Although we know, that $\emptyset \neq \bigcap^g$ Tally-NFA is $\mathrm{NSPACE}(\log n)$-complete in contrast to $\emptyset \neq \bigcap^g$ Tally-DFA, there seem to be no better lower bounds than those given for the deterministic case in Theorem 9.

On the other hand we can give upper bounds analogously to the deterministic case, if we do not use logarithmic space, but polynomial time reductions. Before doing this we prove containment in the appropriate bounded nondeterminism class. In this way the next theorem states that the intersection emptiness problem of $g(n)$ Tally-NFA's can be solved in polynomial time, if we allow a limited number of $g(n) \log n$ nondeterministic steps. For the border case $g(n) = n$ this is the best result possible since in these cases the problem becomes NP-complete, while for $g(n) = 1$ (or $g(n) = \mathrm{const}$) the theorem is not optimal because then the problem gets $\mathrm{NSPACE}(\log n)$ complete, while Theorem 10 gives only the upper bound P.

Theorem 10 $\emptyset \neq \bigcap^{g} \text{Tally-NFA} \in \text{TINOSP}(\text{pol}, \text{pol}, g(n)\log n)$.

Proof: For $\langle A_1, \ldots, A_{g(n)} \rangle$ there exists a word w of length $O(n^{g(n)})$ in $\bigcap_{i=1}^{g(n)} L_{A_i}$, if there exists some word in this intersection. The first step is again to guess the length of such a word and store it within $g(n)\log n$ space. To verify membership of w in $L_{A_1},\ldots,L_{A_{g(n)}}$ we proceed as follows:

Within polynomial time and space we can compute $\delta_{A_i}^{2^k}$, i.e., the transition function of A_i iterated 2^k times, for $k = 0, \ldots, g(n)\log n$. $\delta_{A_i}^{2^k}$ can be computed from $\delta_{A_i}^{2^{k-1}}$ as follows:

$$\delta_{A_i}^{2^k}(q_{i,j}) \quad := \quad \bigcup_{q} \delta_{A_i}^{2^{k-1}}(q), \quad q \in \delta_{A_i}^{2^{k-1}}(q_{i,j})$$

In this way it is not necessary to store the transition function for all possible subsets of states, but polynomial space is sufficient. From this it is easy to compute $\delta_{A_i}^{j}$ for $j \leq n^{g(n)}$ by the same technique. □

The next theorem shows a reduction of $\emptyset \neq \bigcap^{g} \text{Tally-NFA}$ to $\text{NTIME}((g(n))^2 \log^2 n)$. In contrast to Tally-DFA's we have to use a polynomial time reduction, since the computation of the transition table seems to take a lot of space.

Theorem 11 $\emptyset \neq \bigcap^{g} \text{Tally-NFA} \leq_{\text{pol}} \text{NTIME}((g(n))^2 \log^2 n)$.

Proof: As in Theorem 10, the reduction will compute $\delta_{A_i}^{2^k}$. The length of a word w in the intersection is not computed at one blow in advance, but bit for bit in a binary representation.

The actual computation of the time bounded NTM keeps a list of states for all Tally-DFA's $A_1, \ldots, A_{g(n)}$ which is initialized with the initial states. When the kth bit of the binary representation of the length of w is guessed, for each Tally-NFA A_i the following is done:

If $q_{i,j}$ is the current state of A_i held in the list, then $q_{i,j}$ is replaced by some state $q_{i,j'} \in \delta_{A_i}^{2^k}(q_{i,j})$. Which $q_{i,j'}$ is choosen, will be determined nondeterministically which takes $O(\log n)$ nondeterministic steps. The input is accepted, iff the list contains only final states after the computation is finished. □

We close this section by noting, that the restrictions of these unary emptiness problems with respect to deterministic (resp. nondeterministic) finite automata to short words, as it was done in Theorems 5 and 6, always leads to DSPACE($\log n$)-complete (resp. NSPACE($\log n$)-complete) problems regardless of the size of g.

We close this work with the question whether similar results could be obtained, if we would consider alternation instead of nondeterminism.

References

[1] C. Àlvarez, J. Díaz, and J. Torán. Complexity classes with complete problems between P and NP-C. In *Proc. of Conf. on Funfamentals of Computation Theory*, number 380 in LNCS, pages 13–25. Springer, 1989.

[2] J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity Theory I.* Springer, 1988.

[3] J. F. Buss and J. Goldsmith. Nondeterminism within P. In *Proc. of 8th STACS*, number 480 in LNCS, pages 348–359. Springer, 1991.

[4] S. R. Buss. The boolean formula value problem is in ALOGTIME. In *Proc. 19th Ann. ACM Symp. on Theory of Computing*, pages 123–131, 1987.

[5] S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Inform. and Control*, 64:2–22, 1985.

[6] J. Dassow and K.-J. Lange. Computational complexity and hardest languages of automata with abstract storages. In *Proc. of 8th FCT*, number 529 in LNCS, pages 200–209. Springer, 1991.

[7] P. C. Fischer and C. M. R. Kintala. Real-time computations with restricted nondeterminism. *Math. Systems Theory*, 12:219–231, 1979.

[8] Z. Galil. Hierarchies of complete problems. *Acta Inf.*, 6:77–88, 1976.

[9] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, Reading Mass., 1979.

[10] H. Hunt. On the complexity of finite, pushdown, and stack automata. *Math. Systems Theory*, 10:33–52, 1976.

[11] N. Jones. Space-bounded reducibility among combinatorial problems. *J. Comp. System Sci.*, 11:68–85, 1975.

[12] C. M. R. Kintala and P. C. Fischer. Refining nondeterminism in relativezed polynomial-time bounded computations. *SIAM J. Comput.*, 9(1):46–53, 1980.

[13] D. Kozen. Lower bounds for natural proof systems. In *Proc. of 18th FOCS*, pages 254–266, 1977.

[14] R. Ladner and N. Lynch. Relativization of questions about log space computability. *Math. Systems Theory*, 10:19–32, 1976.

[15] I. Niepel. Logarithmisch platzbeschränkte Komplexitätsklassen—Charakterisierung und offene Fragen. Diplomarbeit, University Hamburg, Mar. 1987. (in German).

[16] W. Ruzzo. On uniform circuit complexity. *J. Comp. System Sci.*, 22:365–338, 1981.

[17] L. Stockmeyer and A. Meyer. Word problems requiring exponential time: preliminary report. In *Proc. of the 5th Annual ACM Symposium on Theory of Computing*, pages 1–9, 1973.

[18] K. Wagner and G. Wechsung. *Computational complexity.* Reidel Verlag, Dordrecht and VEB Deutscher Verlag der Wissenschaften, Berlin, 1986.