

INTERACTIVE PROOF SYSTEMS AND ALTERNATING TIME-SPACE COMPLEXITY

LANCE FORTNOW¹

CARSTEN LUND²

UNIVERSITY OF CHICAGO
DEPT. OF COMPUTER SCIENCE
1100 E. 58TH STREET
CHICAGO, IL 60637

Abstract. We show a rough equivalence between alternating time-space complexity and a public-coin interactive proof system with the verifier having a polynomial related time-space complexity. Special cases include

- All of NC has interactive proofs with a log-space polynomial-time public-coin verifier vastly improving the best previous lower bound of $LOGCFL$ for this model [7].
- All languages in P have interactive proofs with a polynomial-time public-coin verifier using $o(\log^2 n)$ space.
- All exponential-time languages have interactive proof systems with public-coin polynomial-space exponential-time verifiers.

To achieve better bounds, we show how to reduce a k -tape alternating Turing machine to a 1-tape alternating Turing machine with only a constant factor increase in time and space.

1. Introduction

In 1981, Chandra, Kozen and Stockmeyer [4] introduced alternating Turing machines, an extension of nondeterministic computation where the Turing

¹Supported by NSF Grant CCR-9009936

²Supported by a fellowship from the University of Århus.

machine can make both existential and universal moves. In 1985, Goldwasser, Micali and Rackoff [10] and Babai [1] introduced interactive proof systems, an extension of nondeterministic computation consisting of two players, an infinitely powerful prover and a probabilistic polynomial-time verifier. The prover will try to convince the verifier of the validity of some statement. However, the verifier does not trust the prover and will only accept if the prover manages to convince the verifier of the validity of the statement.

There are some obvious similarities between alternating Turing machines and interactive proof systems. In fact, Goldwasser and Sipser [11] show the equivalence of interactive proof systems to a Turing machine alternating between nondeterministic and probabilistic moves. However, until recently computer scientists generally believed that alternating Turing machines had far more power than the interactive proof systems.

A series of results by Lund, Fortnow, Karloff and Nisan [14] and Shamir [17] show that the set of languages accepted by an interactive proof system equals the class of languages accepted in deterministic polynomial space. Since Chandra, Kozen and Stockmeyer [4] have shown *PSPACE* equivalent to the languages accepted by a polynomial-time alternating Turing machine, in this case alternating Turing machine and interactive proof systems have identical power.

We generalize the work of [14, 17] to show a broader equivalence between alternating Turing machines and interactive proof systems. We look at time-space complexity, first studied for alternating Turing machines by Ruzzo [16] and for interactive proof systems by Condon [5].

We show a general relationship between time and space bounded alternating Turing machines and time and space bounded verifiers. We show that all languages accepted by an interactive proof system with a $t(n)$ -time $s(n)$ -space bounded verifier can also be accepted by an alternating Turing machine using $t(n) \log t(n)$ -time and $s(n)$ -space. Conversely, we show that an interactive proof system can simulate any $t(n)$ -time $s(n)$ -space alternating Turing machine using a $\text{poly}(n) + \text{poly}(t(n))$ -time and $\text{poly}(s(n))$ -space bounded verifier.

We use this close relationship between alternating Turing machine and interactive proof systems to show a public-coin interactive proof system for all languages in *NC* with a polynomial-time log-space verifier and a proof system for all languages in *P* with a polynomial-time verifier using less than log-squared space. The previous best known result by Fortnow and Sipser [7] shows *LOGCFL* has a public-coin interactive proof system with a polynomial-time log-space verifier. *LOGCFL* consists of all languages log-space reducible to context-free languages and is known to lie in NC^2 [19, 20, 16].

We also use these theorems to get strong relationships between interactive proof systems and deterministic computation similar to the relationships between alternating Turing machine and deterministic computation found in [4].

With the notable exception of Theorem 6.1, most of the results in this paper do not relativize. Fortnow and Sipser [8] have shown an oracle A such that some language in $coNP^A$ does not have interactive proofs relative to A . However our result implies every language in $PSPACE$ has interactive proofs.

2. Background and Definitions

An interactive proof system consists of a prover-verifier pair $P \leftrightarrow V$. The prover and verifier share a reliable communication tape and access to an input tape. The verifier also has access to his own work tapes and a random bit generator such as a fair coin. The prover can be any function from the messages previously sent to the prover to a polynomial-length response. P and V form an interactive protocol for a language L if:

1. If $x \in L$ then $\Pr(P \leftrightarrow V(x) \text{ accepts}) \geq \frac{2}{3}$
2. If $x \notin L$ then for all P^* , $\Pr(P^* \leftrightarrow V(x) \text{ accepts}) \leq \frac{1}{3}$

Goldwasser, Micali and Rackoff [10] and Babai [1] require the verifier computes in probabilistic polynomial time and space. Lund, Fortnow, Karloff and Nisan [14] showed an interactive proof system for every language in the polynomial-time hierarchy. Using their techniques, Shamir [17] showed that the set of languages accepted by these interactive proof systems coincides with the class of languages decidable in deterministic polynomial space. In this paper we will examine the complexity of interactive proof systems with verifiers having differing restrictions on time and space.

In general the verifier may use *private coins* where the prover does not know what the coin tosses were. A *public-coin* interactive proof system allows the prover access to the verifier's coin. Equivalently, we require the verifier's messages to consist of exactly the verifier's coin tosses since the previous round. Goldwasser and Sipser [11] show the class of languages accepted by interactive proofs with a polynomial-time verifier does not depend on whether the verifier uses public or private coins.

However, a difference between private and public coins does seem to hold for time and space bounded verifiers. Condon [5] show that interactive proof systems with private coins and polynomial-time log-space verifiers can simulate any standard interactive protocol and thus accept any $PSPACE$ language.

However, a deterministic polynomial-time Turing machine can simulate any public-coin interactive proof system with a polynomial-time log-space verifier ([5, 7], Corollary 6.2). Thus assuming $P \neq PSPACE$, private coins are strictly more powerful than public coins in an interactive proof system with a time and space bounded verifier. In this paper we study the complexity of the public-coin interactive proof system model.

In this paper we contrast the power of interactive proof systems with alternating Turing machines as developed by Chandra, Kozen and Stockmeyer [4]. An alternating Turing machine is a generalization of a nondeterministic machine where the machine may make both existential and universal choices. A string is accepted by an alternating Turing machine M if there exists a first existential choice such that for all first universal choices there exists a second existential choice ... such that M accepts. See [4] for a complete technical definition.

Let n represent the length of the input string. Let $ATIME(t(n))$ be the set of languages accepted by an alternating Turing machine running in time $O(t(n))$. Let $ASPACE(s(n))$ be the analogous class for space. Chandra, Kozen and Stockmeyer [4] show the following relationships:

- For $s(n) \geq \log n$, $ASPACE(s(n)) = \bigcup_{c>0} DTIME(c^{s(n)})$.
- For $t(n) \geq n$, $ATIME(t(n)) \subseteq DSPACE(t(n)) \subseteq ATIME(t(n)^2)$.

This implies, for example, that $P = ASPACE(\log n)$ and $PSPACE = \bigcup_{k>1} ATIME(n^k)$.

We assume throughout this paper that $t(n), s(n) \geq \log n$, nondecreasing and fully time and space constructible in the following strong sense: There exists a deterministic Turing machine M such that given m written in binary will output the pair $(t(m), 1^{s(m)})$, where $t(m)$ is written in binary. Furthermore M uses $O(t(m))$ time and $O(s(m))$ space. Note most “natural” functions fulfill these conditions. We also assume all inputs are elements of $\{0, 1\}^*$. We say that (t, s) is time-space constructible.

We define the following time and space classes generalizing the *TISP* terminology introduced by Brass and Meyer [3] to describe deterministic computation bounded in both time and space:

A language L is in $ATISP(t(n), s(n))$ if some alternating Turing machine M accepts L and M runs in time $O(t(n))$ and space $O(s(n))$ on every computation path.

A language L is in $IPTISP(t(n), s(n))$ if there exists an public-coin interactive proof for L such that the verifier uses at most $O(t(n))$ time and

$O(s(n))$ space on every computation path with every possible prover. We define $IPTIME(t(n))$ and $IPSPACE(s(n))$ analogously. For $IPSPACE(s(n))$ we restrict the interactive proofs systems to having finite computation paths.

If an alternating Turing machine or a verifier ever enters the same configuration twice then it will have an infinite computation path. Thus we may always assume $s(n) = \Omega(\log t(n))$.

Ruzzo [16] first studied time and space bounded alternating Turing machine complexity showing $ATISP(\log^k n, \log n) = NC^k$ for all $k > 1$ where NC^k is the set of languages accepted by a log-space uniform circuit family of polynomial size and $\log^k n$ depth.

Condon first studied the complexity class $IPTISP(t, s)$ under the name $BC-TIME,SPACE$. In [5] she showed that

$$IPTISP(poly(t(n)), \log t(n)) \subseteq DTIME(poly(t(n))).$$

Fortnow and Sipser [7] studied the class $BPNL = \bigcup_{k>0} IPTISP(n^k, \log n)$. They show $LOGCFL \subseteq BPNL \subseteq P$ where $LOGCFL \subseteq NC^2$ is the class of languages log-space reducible to context-free languages [19, 20].

3. Restricted Alternating Turing Machines

We will use the “random access input” model for an alternating Turing machine (ATM) similar to the one described by Ruzzo [16]. This will allow us to study ATMs which use sublinear time. In our model, the alternating machine M has two special states, q_0 and q_1 . When M enters the state q_j with a value i written in binary on its first work tape, M will accept if the i th bit of the input is j and will reject otherwise. Note that we can simulate arbitrary access to the input by guessing the value of the input and universally verifying that value. We additionally assume that both the verifier and the alternating Turing machine have some constant number k of read-write tapes, each with its own head.

In order to efficiently arithmetize alternating computation, we introduce a special type of alternating Turing machine. It will be a restriction of the model, but we will use the rest of this section to show that it is not a restriction in computational power.

First we will restrict the number of tapes to one and we prove that this does not decrease the computational power.

Paul, Prauss and Reischuk in [15] proved such a theorem for $ATIME(t)$.

THEOREM 3.1. (PAUL, PRAUSS AND REISCHUK [15].) *Let L be a language in $ATIME(t(n))$. There exists a 1-tape alternating Turing machine M such that M works in time $O(t(n))$ and $L(M) = L$.*

We need to extend their result to $ATISP(t, s)$. Recall that $ATISP(t, s)$ is the class of languages that is recognized by alternating Turing machines in time $t(n)$ and space $s(n)$. From their proof it is clear that their simulation uses $O(t(n))$ space. We will present a simulation of a k -tape ATM that works in time $t(n)$ and space $s(n)$ by a 1-tape ATM that works in time $O(t(n))$ and space $O(s(n))$. In our proof we will use both their result and the ideas of their proof.

THEOREM 3.2. *Let $L \in ATISP(t(n), s(n))$. There exists a 1-tape alternating Turing machine M that works in time $O(t(n))$ and space $O(s(n))$ and $L(M) = L$.*

PROOF. Let N be a k -tape ATM that recognizes L such that N works in time $O(t(n))$ and space $O(s(n))$. We construct M such that it simulates N in *phases* corresponding to time blocks of size $s(n)$. At the beginning of each phase M will have the contents of N 's tapes stored on a track on its work tape. M will start each phase by guessing the next $s(n)$ *displays* of N on a second track of its work tape. The display of N at time i consists of the state and the content of each of the k cells that N scans at time i . M will be in a universal (resp. existential) state if N is in a universal (resp. existential) state. It will next on a third track existentially guess the content of N 's tapes after the phase. Note that if the displays correspond to a valid computation path of N then there exist such tape contents, which is unique.

Thereafter M checks the validity of its guesses. Observe that checking the validity of the guesses can easily be done on a $k + 1$ tape deterministic Turing machine M_1 in time $O(s(n))$. Hence because of Theorem 3.1 M can simulate M_1 using one tape in time $O(s(n))$. If the guesses correspond to a computation path in N then M starts a new phase. If not, it has to figure out which type of state it was in when it made the first wrong guess. Note that M has to reject (resp. accept) if it guessed wrong in an existential (resp. universal) state. But a $k + 1$ tape deterministic Turing machine M_2 can in time $O(s(n))$ easily find the type of the state, where the first wrong guess was made. M can simulate M_2 in time $O(s(n))$ using Theorem 3.1. Furthermore, if at some point N accepts or rejects, M does the same. It is easy to see that $L(M) = L(N)$ and that M works in time $O(t(n))$ and space $O(s(n))$. \square

We will restrict our model even further. We will let the ATM first make an existential move consisting only of two possible moves followed by a universal move of again two possible moves and so on. This will make the computation tree a binary tree and with alternating levels of AND gates and OR gates. Furthermore we will assume that all computation paths have even length. It is easy to see that given an arbitrary ATM M we can construct an ATM N that has a computation tree as described above and that N works in time and space proportional to the time and space used by M .

Hence the machines we will consider are *restricted* Alternating Turing machines M such that

- M has only one tape, which at the start of the computation contains the binary representation of $n = |x|$.
- M 's computation tree is a binary tree with alternating levels of AND gates and OR gates and all computation paths have even length.
- On each computation path M will only read one input bit and will do that at the very end. A computation path will accept or reject depending on the i th bit of the input, if the binary representation of i is stored in the first $\lceil \log n \rceil$ cells on the work tape at the end of the computation, otherwise it will reject.

4. Arithmetization of Alternating Computation

The proof of our main result extends the algebraic techniques, which was used in the recent results on the power of interactive proof systems [14, 17].

Let $L \in ATISP(t, s)$ and let M be a restricted ATM such that $L(M) = L$. We will assume without loss of generality that M works in time $t(n)$ and uses $s(n)$ space. Let Δ be the work alphabet and let Q be the set of M 's states. Given an input string x , we will define an arithmetic expression E_x , such that the value of E_x determines if $x \in L$. We construct E_x in this section and in the next section we will show how an interactive proof system can verify the value of E_x .

A *Boolean function* in n variables is a function $\{0, 1\}^n \rightarrow \{0, 1\}$. A polynomial $g(x_1, x_2, \dots, x_n)$ (over some field) *interpolates* a Boolean function f on n variables if for all $(0,1)$ -substitutions the (Boolean) value of f and the (arithmetic) value of g agree. A polynomial is *multilinear* if it is linear in every variable.

A *Boolean expression* is a well-formed expression built from the constants 0,1 and variable symbols using the operations $\wedge, \vee, \neg, \forall$ and \exists . A *Boolean formula* is a Boolean expression using only the operations \wedge, \vee and \neg . A Boolean expression represents a Boolean function in the obvious sense.

An *arithmetic expression* is a well-formed expression built from the constants 0,1 and variable symbols using the operations $+, -, \times, \prod$ and \sum . An *arithmetic formula* is an arithmetic expression using only the operations $+, -$ and \times . An arithmetic expression represents a polynomial function in the obvious sense over any field.

An expression correspond to a labeled tree where the leaves are labeled by constants or variable symbols and the internal nodes are labeled by operations. The *size* of a expression is the number of operations used to build the expression. Hence the size is equal to the number of internal node in the tree corresponding to the expression. The *depth* of a expression is the length of the longest path from the root to a leaf in the tree corresponding to the expression.

First we need a proposition stating that every Boolean function is interpolated by a multilinear polynomial.

PROPOSITION 4.1. *Given a Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ there exists a unique multilinear function $g : \mathbf{Z}^n \rightarrow \mathbf{Z}$ interpolating f .*

PROOF. Left to the reader. \square

The basic idea in the arithmetization is to first construct a Boolean predicate that is true if and only if $x \in L$. Thereafter we arithmetize the Boolean predicate.

Let $\varphi_i(I, x)$ be the predicate that is true if and only if from the configuration described by the ID I , in at most $2i$ steps M will accept x . We will assume that I describes M to be in an existential state. Let $M(I)$ denote the machine M in the configuration described by I . Because of the simple structure of the computation of the restricted machine M it is clear for $i > 0$ that $M(I)$ accepts x in at most $2i$ steps if

- $M(I)$ makes an existential guess a followed by a universal guess b and ends up having an ID I' such that $M(I')$ accepts x in at most $2(i-1)$ steps.

We will make the assumption that when M enters a final configuration it will stay in this configuration indefinitely. For $i = 0$, $M(I)$ accepts x if and only if I describes an accepting configuration. This gives an inductive definition of

$\varphi_i(I, x)$:

$$\varphi_i(I, x) = \begin{cases} g_x(I) & \text{if } i = 0 \\ \exists a \forall b \exists I' : f(I, I', a, b) \wedge \varphi_{i-1}(I', x) & \text{otherwise} \end{cases}$$

where $f(I, I', a, b)$ is the predicate stating that $M(I)$, on input x , on an existential guess a and a universal guess b , gets into the configuration described by I' . The predicate $g_x(I)$ states that M accepts input x if in configuration I .

If $N = \lceil t(n)/2 \rceil$ and I_0 is the ID describing the start configuration then clearly

$$\varphi_N(I_0, x) = 1 \iff M \text{ accepts } x.$$

We will extend the ideas by Shamir [17] and Babai-Fortnow [2] to arithmetize $\varphi_N(I_0, x)$. The technique is polynomial extrapolation of truth values. We will given a Boolean φ expression construct an arithmetic expression A_φ . The following table inductively defines A_φ .

φ	A_φ
0	0
1	1
x_i	x_i
$\neg \varphi'$	$1 - A_{\varphi'}$
$\varphi' \wedge \varphi''$	$A_{\varphi'} A_{\varphi''}$
$\varphi' \vee \varphi''$	$1 - (1 - A_{\varphi'})(1 - A_{\varphi''})$
$\forall x : \varphi'(x)$	$\prod_{x=0}^1 A_{\varphi'}(x)$
$\exists x : \varphi'(x)$	$\prod_{x=0}^1 A_{\varphi'}(x)$

where $\prod_{x=0}^1 A_{\varphi'}(x)$ is a short hand for $1 - \prod_{x=0}^1 (1 - A_{\varphi'}(x))$.

In what follows we will use these rule except with a slight modification in the case of $\varphi = \exists x : \varphi'(x)$. In some cases we will let $A_\varphi = \sum_{x=0}^1 A_{\varphi'}(x)$. This will be in the case where we know that $\varphi'(x)$ is true, for at most one value of x . The advantage is that the degree of the polynomial A_φ is at most the degree of $A_{\varphi'}$ whereas in the original case the degree double. It is crucial to keep the degree low as we will see later.

Using these rule we get that

$$A_\varphi = \begin{cases} 1 & \text{if } \varphi \text{ is true} \\ 0 & \text{otherwise.} \end{cases}$$

This can easily be proven by induction.

First we will need an encoding of IDs in order to arithmetize $f(I, I', a, b)$ and $g_x(I)$.

Our encoding of IDs will be a compact version of a binary encoding. The binary encoding encodes I as a tuple $(q, c_1, c_2, \dots, c_{s(n)}, h_1, h_2, \dots, h_{s(n)})$, where

- $q = (q_1, q_2, \dots, q_{k'})$ is a binary encoding of the state of M , where $k' = \lceil \log_2(|Q|) \rceil$.
- $c_i = (c_{i1}, c_{i2}, \dots, c_{ik})$ is a binary encoding of the content of the i th cell, where $k = \lceil \log_2(|\Delta|) \rceil$.
- h_i is 1 if and only if the head of M is scanning the i th cell.

We will first describe how to arithmetize $\varphi_N(I_0, x)$ using the binary encoding and later we will extend the arithmetization to a more compact encoding.

In order to arithmetize the computation of M we will first assume that the head is scanning the i th cell. Let $I = (q, c_1, \dots, h_{s(n)})$ and $I' = (q', c'_1, \dots, h'_{s(n)})$. Given the head position it is locally decidable if I' follows I on the guesses a and b . Define

$$\varphi_{mi} := [c'_m \text{ is correct given that the head scans cell } i]$$

$$\eta_{mi} := [h'_m \text{ is correct given that the head scans cell } i]$$

$$\psi_i := [q' \text{ is correct given that the head scans cell } i]$$

We will first look at which variables the above predicates depend on.

- φ_{mi} depends only on c_m and c'_m if $m \notin \{i-1, i, i+1\}$, since in two steps M can only change the content of cells $i-1, i, i+1$. For $m \in \{i-1, i, i+1\}$, φ_{mi} depends on $q, c_{i-1}, c_i, c_{i+1}, a, b$ and c'_m .
- η_{mi} depends only on h'_m if $m \notin \{i-2, i-1, i, i+1, i+2\}$, since in two steps M can only move its head two positions. For $m \in \{i-2, i-1, i, i+1, i+2\}$, η_{mi} depends on $q, c_{i-1}, c_i, c_{i+1}, a, b$ and h'_m .
- ψ_i depends on the variables $q, c_{i-1}, c_i, c_{i+1}, a, b$ and q' .

Let f_i be the predicate that is true if and only if I' follows I on the guesses a and b , given that the head is scanning the i th cell. From the above definitions

we can write down a simple Boolean formula for f_i . We get that f_i is true if and only if

$$\psi_i \wedge \bigwedge_{m=1}^{s(n)} (\varphi_{mi} \wedge \eta_{mi})$$

is true.

But since h_i describes if the head is at cell i in I , f can be written down as

$$\bigvee_{i=1}^{s(n)} (h_i \wedge f_i).$$

Given this description of f , it is now straightforward to arithmetize f . By Proposition 4.1 any predicate γ can be interpolated by a multilinear polynomial P_γ . If γ only depends on a fixed number of variables then P_γ can be computed by a formula F_γ of fixed size. Hence we obtain formulas $F_{\varphi_{mi}}$, $F_{\eta_{mi}}$ and F_{ψ_i} that compute multilinear polynomials that interpolate φ_{mi} , η_{mi} and ψ_i respectively.

This gives arithmetic formulas that compute polynomials that interpolate f_i and f :

$$F_{f_i} := F_{\psi_i} \prod_{m=1}^{s(n)} F_{\varphi_{mi}} F_{\eta_{mi}}$$

$$F_f := \sum_{i=1}^{s(n)} h_i F_{f_i}$$

LEMMA 4.2. *For any $I, I' \in \{0, 1\}^{(k+1)s(n)+k'}$, where I is an encoding of a valid ID, and for any $a, b \in \{0, 1\}$,*

$$F_f(I, I', a, b) := \begin{cases} 1 & \text{if } f(I, I', a, b) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore the degree of any variable in F_f is at most 9, $\text{size}(F_f) = O(s^2(n))$ and $\text{depth}(F_f) = O(\log s(n))$.

PROOF. That F_f interpolates f is clear when we observe that for at most one i , $h_i \wedge f_i$ is true. The reason being that for at most one i , h_i is true, since I is an encoding of a valid ID. Thus at most one of the summands of F_f is one.

To find the degree of, say, q_1 , note that in f_i , it is only $\psi_i, \varphi_{i-1,i}, \varphi_{i,i}, \varphi_{i+1,i}, \eta_{i-2,i}, \eta_{i-1,i}, \eta_{i,i}, \eta_{i+1,i}$ and $\eta_{i+2,i}$ that depend on q_1 . Hence q_1 will only be a variable in the corresponding 9 formulas, each of which is multilinear. Hence the degree of q_1 is at most 9 in F_{f_i} and therefore in F_f . The following table gives a complete description of the bounds on the degrees for every variable.

	q_j	q'_j	c_{mj}	c'_{mj}	h_m	h'_m	a	b
F_{f_3}	9	1	9	1	0	1	9	9
F_f	9	1	9	1	1	1	9	9

This completes the proof. \square

To arithmetize g_x observe that, because M is restricted, g_x depends only on $c_1, c_2, \dots, c_{\lceil \log n \rceil}, q$ and x . In order for I to be an accepting ID $c_1, c_2, \dots, c_{\lceil \log n \rceil}$ should be the binary representation of some number $j \in \{1, 2, \dots, n\}$ and furthermore, the input bit x_j should be 0 or 1 depending on q . So for each j and $l \in \{1, 2, \dots, \lceil \log n \rceil\}$ we define $\sigma_{jl}(c_l)$ to be true if and only if c_l encodes the l th bit in the binary representation of j . Furthermore we define $\rho_j(q, x_j)$ to be true if and only if in state q , M immediately accepts if the j th input is x_j . Hence

$$g_x = \bigvee_{j=1}^n \left(\bigwedge_{l=1}^{\lceil \log n \rceil} \sigma_{jl}(c_l) \right) \wedge \rho_j(q, x_j).$$

This gives us the following arithmetic formula,

$$F_{g_x} := \sum_{j=1}^n \left(\prod_{l=1}^{\lceil \log n \rceil} F_{\sigma_{jl}}(c_l) \right) F_{\rho_j}(q, x_j),$$

where $F_{\sigma_{jl}}$ and F_{ρ_j} are formulas that compute multilinear polynomials that interpolate σ_{jl} and ρ_j respectively.

LEMMA 4.3. *For any $I \in \{0, 1\}^{(k+1)s(n)+k'}$, where I is an encoding of a valid ID,*

$$F_{g_x}(I) = \begin{cases} 1 & \text{if } g_x(I) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore the degree of any variable in F_{g_x} is at most 1, $\text{size}(F_{g_x}) = O(n \log n)$ and $\text{depth}(F_{g_x}) = O(\log n)$.

PROOF. Clear since there exists at most one j such that

$$\left(\bigwedge_{l=1}^{\lceil \log n \rceil} \sigma_{jl}(c_l) \right) \wedge \rho_j(q, x_j)$$

is true. \square

We improve the space complexity of the encoding of IDs by using a more compact encoding. Choose an $\epsilon > 0$ and encode $m = \lfloor \frac{\epsilon}{2} \log s(n) \rfloor$ binary

symbols $\alpha_0, \alpha_1, \dots, \alpha_{m-1}$ into a new symbol $\alpha \in \{0, 1, \dots, 2^m - 1\}$, just by letting $\alpha = \sum_{i=0}^{m-1} \alpha_i 2^i$. Let $X = \{0, 1, \dots, 2^m - 1\}$. To decode the new symbol define $D_i : X \rightarrow \{0, 1\}$ that maps an m -bit number α into the i th bit of α . The decoding is obtained, using Lagrange interpolation, by the following formula.

$$F_{D_i}(x) = \sum_{u \in X} D_i(u) \prod_{v \neq u} \frac{(x - v)}{(u - v)}$$

The degree of the polynomial computed by F_{D_i} is $2^m - 1$, and $\text{size}(F_{D_i}) = O(2^{2m})$ and $\text{depth}(F_{D_i}) = O(m)$. Define

$$F(y_1, y_2, \dots, y_{n'}, y'_1, y'_2, \dots, y'_{n'}, a, b) := F_f(F_{D_1}(y_1), F_{D_2}(y_1), \dots, F_{D_m}(y_1), F_{D_1}(y_2), \dots, F_{D_m}(y'_{n'}), a, b)$$

where $n' = \lceil \frac{(k+1)s(n)+k'}{m} \rceil$. In other words we modify the formula F_f . For each leaf l containing a variable z that encodes part of an ID, we are replacing l with the subformula that decodes z from the appropriate variable of F . For example, if z is z_1 , the first binary variable that encodes I , then the subformula is $F_{D_1}(y_1)$, since y_1 encodes z_1 .

LEMMA 4.4. *For any $I, I' \in X^{n'}$, where I is an encoding of a valid ID, and for any $a, b \in \{0, 1\}$,*

$$F(I, I', a, b) = \begin{cases} 1 & \text{if } f(I, I', a, b) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore the degree of any variable in F is at most $9(2^m - 1) < 9s^{\epsilon/2}(n)$, $\text{size}(F) = O(2^{2m} s^2(n)) = O(s^{2+\epsilon}(n))$ and $\text{depth}(F) = O(\log s(n))$.

PROOF. Follows from Lemma 4.2. \square

We define G_x in a way similar to that of the definition of F_{g_x} .

$$G_x(y_1, \dots, y_{n'}) := F_{g_x}(F_{D_1}(y_1), F_{D_2}(y_1), \dots, F_{D_m}(y_1), F_{D_1}(y_2), \dots, F_{D_m}(y_{n'}))$$

LEMMA 4.5. *For any $I \in X^{n'}$, where I is an encoding of a valid ID,*

$$G_x(I) = \begin{cases} 1 & \text{if } g_x(I) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

Furthermore the degree of any variable in G_x is at most $(2^m - 1) < s^{\epsilon/2}(n)$, $\text{size}(G_x) = O(2^{2m} n \log n) = O(s^\epsilon(n) n \log n)$ and $\text{depth}(G_x) = O(\log s(n) + \log n)$.

PROOF. Clear from Lemma 4.3. \square

With the arithmetization of f and g_x , we get an arithmetization of $\varphi_i(I, x)$ by inductively defining

$$A_i(I, x) = \begin{cases} G_x(I) & \text{if } i = 0, \\ \prod_{a=0}^1 \prod_{b=0}^1 \sum_{I' \in X^{n'}} F(I, I', a, b) \cdot A_{i-1}(I', x) & \text{otherwise,} \end{cases}$$

where $X = \{0, 1, \dots, 2^m - 1\}$. Remember that $\prod_{x=0}^1 q(x) = 1 - \prod_{x=0}^1 (1 - q(x))$.

LEMMA 4.6. *If $I_0 = (y_1, y_2, \dots, y_{n'}) \in X^{n'}$ is the encoding of M 's starting configuration then*

$$A_N(I_0, x) = \begin{cases} 1 & \text{if } x \in L, \\ 0 & \text{otherwise.} \end{cases}$$

PROOF. We prove by induction that for all i and for all encodings I of valid IDs,

$$A_i(I, x) = \begin{cases} 1 & \text{if } \varphi_i(I, x) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

For $i = 0$ this follows from Lemma 4.5.

For $i > 0$, observe that there is one I' such that $f(I, I', a, b)$ is true. Observe that both I and I' are encodings of valid IDs. Now it follows from Lemma 4.4 and the inductive hypothesis that

$$A_i(I, x) = \begin{cases} 1 & \text{if } \varphi_i(I, x) \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}$$

\square

5. Interactive Proof Systems for $ATISP(t, s)$

The verifier has to check that $A_N(I_0, x) = 1$. The protocol starts by the prover sending the verifier a prime $p \in [30dN(n' + 2), 60dN(n' + 2)]$, where d is the maximal degree of any variable in F and G_x (Lemma 4.4 and Lemma 4.5 shows that $d \leq 9s^{\epsilon/2}(n)$). The verifier thereafter tests that the number the prover sent is indeed a prime using the primality test of Solovay-Strassen [18], such that the verifier will catch the prover with probability at least $1/6$ if the prover tries to cheat in this initial stage of the protocol.

The verifier can simplify the arithmetic expression $A_N(I_0, x)$ by the technique of Lund-Fortnow-Karloff-Nisan [14]. The verifier will be working over the

finite field \mathbf{F}_p with p elements. At any point in time the verifier will know an arithmetic expression $A(y_1, y_2, \dots, y_k)$ which is a suffix of $A_N(I_0, x)$ and values $\alpha_1, \alpha_2, \dots, \alpha_k, \beta \in \mathbf{F}_p$ and the verifier has to verify that $A(\alpha_1, \alpha_2, \dots, \alpha_k) = \beta$. There are three different types of A .

1. $A(y_1, \dots, y_k) = \bigodot_{z \in Y} B(y_1, \dots, y_k, z)$, where $\bigodot \in \{\sum, \prod, \coprod\}$ and $Y = \{0, 1\}$ or $Y = X$. Observe that $B(\alpha_1, \dots, \alpha_k, z)$ is a polynomial $q(z)$ over \mathbf{F}_p . By inspecting the definition of $A_N(I_0, x)$ we see that the degree of $q(z)$ is at most $5d$. So the protocol is

- P** \rightarrow **V**: $q'(z)$ a at most $5d$ degree polynomial over \mathbf{F}_p .
- V**: Check $\bigodot_{z \in Y} q'(z) = \beta$.
- V**: Choose randomly and uniformly $\gamma \in \mathbf{F}_p$.
- V**: Continue the protocol and check that $B(\alpha_1, \alpha_2, \dots, \alpha_k, \gamma) = q'(\gamma)$.

Now the proof of correctness for this step is that if $A(\alpha_1, \dots, \alpha_k) \neq \beta$ then $q'(z) \neq B(\alpha_1, \dots, \alpha_k, z)$ and hence with probability at most $5d/p$ we have $B(\alpha_1, \dots, \alpha_k, \gamma) = q'(\gamma)$. We say that if this happens then the cheating prover *succeeds* in this step.

2. $A(y_1, y_2, \dots, y_k) = F(y_1, y_2, \dots, y_{2n'+2})A_i(y_{n'+1}, y_{i+2}, \dots, y_{2n'})$.
 - V**: Calculate $\delta = F(\alpha_1, \alpha_2, \dots, \alpha_{2n'+2})$. If $\delta = 0$ then accept if $\beta = 0$ otherwise reject. Otherwise continue by checking that $A_i(y_{n'+1}, y_{i+2}, \dots, y_{2n'}) = \delta^{-1}\beta$.
3. $A(y_1, y_2, \dots, y_k) = G_x(y_1, y_2, \dots, y_{n'})$.
 - V**: Calculate $\delta = G_x(\alpha_1, \alpha_2, \dots, \alpha_{n'})$. Accept if $\beta = \delta$ otherwise reject.

We can then prove that the above protocol recognizes L .

LEMMA 5.1. *The protocol above satisfies the following statements:*

- i. *If $x \in L$ then there exists a prover such that the verifier always accepts.*
- ii. *If $x \notin L$ then for all provers, the verifier accepts with probability at most $\frac{1}{3}$.*
- iii. *The verifier works in time*

$$O((t(n)s^{2+\epsilon}(n) + s^\epsilon(n)n \log n) \log^2 t(n))$$

and uses space

$$O\left(\frac{s(n) \log p}{\log s(n)}\right) = O\left(\frac{s(n) \log t(n)}{\log s(n)}\right).$$

PROOF.

- i. The prover will always be able to make $A(\alpha_1, \alpha_2, \dots, \alpha_k) = \beta$. Note that this is the case from the start since $A_N(I_0, x) = 1$. In each elimination step the prover accomplish this by always letting $q'(z) = B(\alpha_1, \alpha_2, \dots, \alpha_k, z)$.
- ii. The cheating prover has two opportunities to make the verifier accept. First it can choose a composite number for p . Since the verifier is doing the primality test the prover will be caught with probability at least $5/6$. Secondly the prover can succeed in one of the elimination steps. Note that in each elimination step the probability that the prover succeeds in that step is at most $\frac{5d}{p}$ assuming that p is a prime. Since the prover has only $N(n' + 2)$ chances to succeed and since $p \geq 30dN(n' + 2)$ the probability that the prover will succeed in any elimination step, given that p is a prime, is at most $\frac{5dN(n'+2)}{p} < \frac{1}{6}$. Hence in all this implies that the verifier accepts x with probability at most $1/3$.
- iii. The verifier uses $O(N((3+3+n'(|X|+1))T_{eval}+T_F)+T_{G_x}))$ additions and multiplications in \mathbf{F}_p , where T_{eval} is the number of operations to evaluate a polynomial of degree $5d$ at one point, T_F is the number of operations to evaluate F at one point and T_{G_x} is the number of operations to evaluate G_x at one point.

By Horner's Method $T_{eval} = O(5d)$. The the number of arithmetic operations needed to evaluate the arithmetic formula of size S is clearly $O(S)$. Hence the number of additions and multiplications performed by the verifier is

$$O(t(n)s^{2+\epsilon}(n) + s^\epsilon(n)n \log n).$$

Furthermore the verifier computes $O(t(n))$ inverses.

The time for a TM to perform one addition in \mathbf{F}_p is $O(\log p)$ and multiplications can be done in time $O(\log^2 p)$. To compute an inverse using the extended GCD algorithm takes time $O(\log^3 p)$. The primality test for p takes time $O(\log^3 p)$.

Hence we infer that the verifier works in time

$$O((t(n)s^{2+\epsilon}(n) + s^\epsilon(n)n \log n) \log^2 t(n))$$

The verifier uses space for two independent reasons. She is storing $O(n')$ field elements and she uses space to evaluate the polynomials and the formulas. It is clear that a formula can be evaluated using only a number of registers proportional to the depth of the formula. Hence the verifier needs $O(n') = O(s(n)/\log s(n))$ registers each containing an element from \mathbf{F}_p . Hence the verifier uses

$$O\left(\frac{s(n) \log p}{\log s(n)}\right) = O\left(\frac{s(n) \log t(n)}{\log s(n)}\right)$$

space. \square

This gives the main result of this paper. Recall that $IPTISP(t, s)$ is the class of languages for which there exists an interactive proof system with a public coin verifier that works in time $O(t(n))$ and space $O(s(n))$.

THEOREM 5.2. *Given that (t, s) is fully time-space constructible, if*

$$L \in ATISP(t(n), s(n)),$$

then L belongs to

$$\bigcap_{\epsilon > 0} IPTISP((s^2(n)t(n) + n \log n)s^\epsilon(n) \log^2 t(n), s(n) \log t(n) / \log s(n)).$$

COROLLARY 5.3. *Every language in P has a public-coin interactive proof system with a polynomial-time verifier using $O(\log^2(n) / \log \log(n))$ space.*

PROOF. Let L be a language in P . Chandra, Kozen and Stockmeyer [4] prove the existence of an alternating polynomial-time log-space Turing machine M that accepts L . \square

COROLLARY 5.4. *Every language in $NC = \bigcup_k NC^k$ has a public-coin interactive proof system with a verifier using $O(\log(n))$ space and $O(n \log^2 n)$ time.*

PROOF. Ruzzo [16] shows that any language in NC can be accepted by an alternating Turing machine using poly-log time and log space. Hence if $L \in NC$, then there exists a constant k such that

$$\begin{aligned} L &\in ATISP(\log^k n, \log n) \\ &\subseteq \bigcap_{\epsilon > 0} IPTISP(n(\log^{1+\epsilon} n)(\log \log n)^2, \log n) \\ &\subseteq IPTISP(n \log^2 n, \log n). \square \end{aligned}$$

COROLLARY 5.5. *If (t, s) is fully time-space constructible, then*

$$ATISP(t(n), s(n)) \subseteq \bigcap_{\epsilon > 0} IPTISP(n^{1+\epsilon} + t^{3+\epsilon}(n), s^2(n)/\log s(n)).$$

PROOF. This follows from Theorem 5.2, since $t(n) \geq s(n)$ and $s(n) \geq \log t(n)$. \square

6. Alternating Turing Machines for $IPTISP(t, s)$

THEOREM 6.1. *Let (t, s) be fully time-space constructible. Then*

$$IPTISP(t(n), s(n)) \subseteq ATISP(t(n) \log t(n), s(n)).$$

PROOF. Let L have a public-coin interactive proof system using time $t(n)$ and space $s(n)$. We can assume without loss of generality that the protocol consists of exactly $t(n)$ rounds of the verifier sending a single coin toss to the prover followed by the prover sending back a one-bit response.

From any configuration c , the probability that the verifier accepts, starting in configuration c , must be $\frac{v_c}{2^{t(n)}}$ for some integer v_c , with $0 \leq v_c \leq 2^{t(n)}$. Let v_c be the *value* of a configuration c . The value of an accepting configuration is 1 and the value of a rejecting configuration is 0.

If c is the configuration immediately before a prover's message and if a prover response of 0 causes the verifier to enter configuration c_0 and a response of 1 causes the verifier to enter configuration c_1 , then $v_c = \max(v_{c_0}, v_{c_1})$. If c is the configuration immediately before a coin toss, c_0 is the configuration the verifier enters after tossing heads, and c_1 is the configuration the verifier enters after tossing tails, then $v_c = (v_{c_0} + v_{c_1})/2$.

An alternating machine to accept L can work as follows: First existentially guess the value of the initial configuration and then verify its guess.

To verify its guess, we notice that maximum, addition (using carry look-ahead) and division by 2 on $t(n)$ -bit numbers has a uniform space $O(\log t(n))$ circuit of depth $O(\log t(n))$. So the value of the initial configuration can be calculated by a uniform (space $O(s(n))$) circuit of depth $O(t(n) \log t(n))$. By a result by Ruzzo [16] an alternating Turing machine can evaluate each bit of the value of the initial configuration in time $O(t(n) \log t(n))$ and space $O(s(n))$. \square

Condon [5] and Fortnow and Sipser [7] independently proved the following fact, which follows from Theorem 6.1.

COROLLARY 6.2. *A deterministic polynomial-time Turing machine can recognize any language accepted by a public-coin interactive proof system with a verifier using logarithmic space and polynomial time.*

PROOF. $\bigcup_{k>0} ATISP(n^k, \log n) = ASPACE(\log n) = P$ [4] \square

7. A Hierarchy for $IPTISP(t, s)$

Theorem 3.2 gives a tight hierarchy for $ATISP(t, s)$.

THEOREM 7.1. *Given (t_1, s_1) and (t_2, s_2) fully time-space constructible pairs of functions,*

$$ATISP(t_1, s_1) \subsetneq ATISP(t_2, s_2)$$

if

$$t_1(n) = o(t_2(n)) \text{ and } s_1(n) = o(s_2(n))$$

PROOF. Let M_1, M_2, \dots be an enumeration of 1-tape alternating Turing machines, such that every Turing machine has arbitrarily long encodings. We construct an alternating Turing machine M that uses time $O(t_2(n))$ and space $O(s_2(n))$, and that recognizes a language not in $ATISP(t_1, s_1)$.

The idea is that M tries to diagonalize against all the machines in the enumeration. It will succeed against all machines using time $O(t_1(n))$ and space $O(s_1(n))$.

On input x , M simulates M_x on input x , in the following way. If M_x makes an existential guess then M will make a universal guess and vice versa. Doing the simulation M keeps track on the time and the space it is using. If on some computation path M discover that it has used more than $t_2(n)$ time or $s_2(n)$ space and it will halt and reject the input. Otherwise the simulated machine M_x halts and M will accept if and only if M_x rejects the input x . Note that if on all computation paths the simulation succeeds then M accepts x if and only if M_x rejects.

Clearly M uses $O(t_2(n))$ time and $O(s_2(n))$ space, since (t_2, s_2) are time-space constructible. Hence $L(M) \in ATISP(t_2, s_2)$.

Assume that $L(M) \in ATISP(t_1, s_1)$. Hence we have an alternating Turing machine M' that recognizes $L(M)$ and it works in time $ct_1(n)$ and space $cs_1(n)$, for some constant c . Note that given M' there exists a constant c' such that M simulates M' on an encoding of M' with a slow down of at most c' and

uses only a factor of c' more space than M' . Since there are arbitrarily long encodings of M' let x be an encoding of M' such that

$$cc't_1(n) \leq t_2(n) \text{ and } cc's_1(n) \leq s_2(n).$$

Now M accepts x if and only if M_x rejects x . This contradicts that $L(M) = L(M')$. \square

We get a hierarchy theorem for $IPTISP(t, s)$, as a consequence of our correspondence between $ATISP(t, s)$ and $IPTISP(t, s)$.

THEOREM 7.2. *Given (t_1, s_1) and (t_2, s_2) , fully time-space constructible pairs of functions such that $t_1, t_2 \geq n$ and $s_1, s_2 \geq \log n$,*

$$IPTISP(t_1(n), s_1(n)) \subsetneq IPTISP(t_2(n), s_2(n))$$

if for some $\epsilon > 0$

$$t_1^{3+\epsilon}(n) = o(t_2(n)) \text{ and } s_1^2(n) = o(s_2(n)).$$

PROOF.

$$\begin{aligned} IPTISP(t_1(n), s_1(n)) &\subseteq ATISP(t_1(n) \log t_1(n), s_1(n)) \\ &\subsetneq ATISP(t_2(n)^{\frac{1}{3+\epsilon/2}}, s_2(n)^{\frac{1}{2}}) \\ &\subseteq IPTISP(n^{1+\epsilon/2} + t_2(n), s_2(n)) \\ &= IPTISP(t_2(n), s_2(n)). \end{aligned}$$

The first containment is from Theorem 6.1, the proper containment is Theorem 7.1 and the last containment is from Corollary 5.5. \square

As a corollary we get that public coin interactive proof systems with linear time verifiers can not recognize all of IP . This should be contrasted with the result by Fortnow and Sipser [9] that for probabilistic computation there exists an oracle A such that $BPTIME(n)^A$ contains BPP^A . Furthermore, Theorem 7.2 gives a tighter hierarchy for time and space.

COROLLARY 7.3. *For all reals $1 \leq r < s$,*

$$IPTIME(n^r) \subsetneq IPTIME(n^s),$$

$$IPSPACE(n^r) \subsetneq IPSPACE(n^s)$$

and

$$IPSPACE(\log^r n) \subsetneq IPSPACE(\log^s n).$$

PROOF. Given Theorem 7.2 the proof is similar to the proof of similar results for $NSPACE(s)$ by Ibarra [13] (see Theorem 12.12 in [12]). \square

8. Interactive Proof Systems for Deterministic Computation

Corollaries 5.3 and 5.4 exhibit interactive proof systems with verifiers having low time-space complexity for P and NC . We can use Theorem 6.1 and Corollary 5.5 combined with the relationships in [4] described in Section 2 to prove more general relationships.

COROLLARY 8.1. For $t(n) \geq n, s(n) \geq \log n$,

- $\bigcup_{k>0} IPTISP(t(n)^k, t(n)^k) = \bigcup_{k>0} DSPACE(t(n)^k)$.
- $\bigcup_{k>0} IPTISP(2^{s(n)^k}, s(n)^k) = \bigcup_{k>0} DTIME(2^{s(n)^k})$.

From this we get several consequences including:

1. An interactive protocol with a verifier using poly-log space and running in quasi-polynomial ($2^{\text{poly-log}(n)}$) time accepts the same set of languages as a deterministic Turing machine running in quasi-polynomial time.
2. A public coin interactive protocol with a verifier running in polynomial time and space accepts exactly the same set of languages as a deterministic machine using polynomial space.
3. An interactive protocol with a verifier using polynomial space and exponential time accepts exactly the same set of languages as are deterministically recognizable in exponential time.
4. An interactive protocol with a verifier using exponential time and space can accept all languages deterministically recognizable in exponential space.

The second consequence is equivalent to Shamir's result that $IP = PSPACE$.

References

- [1] L. Babai. Trading group theory for randomness. In *Proc. of the 17th ACM Symp. on the Theory of Computing*, pages 421–429, 1985.

- [2] L. Babai and L. Fortnow. Arithmetization: a new method in structural complexity theory. *Computational Complexity*, 1:41–66, 1991.
- [3] A. R. Brass and A. R. Meyer. On time-space classes and their relation to the theory of real addition. *Theoretical Computer Science*, 11:59–69, 1980.
- [4] A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. of the ACM*, 28(1):114–133, 1981.
- [5] A. Condon. Space bounded probabilistic game automata. In *Proc. of the 3rd Conference on Structure in Complexity Theory*, pages 162–174, 1988.
- [6] L. Fortnow. *Complexity-theoretic aspects of interactive proof systems*. PhD thesis, Massachusetts Institute of Technology, Laboratory for Computer Science, 1989. Tech Report MIT/LCS/TR-447.
- [7] L. Fortnow and M. Sipser. Interactive proof systems with a log-space verifier. Manuscript. Later version appears in [6].
- [8] L. Fortnow and M. Sipser. Are there interactive protocols for co-NP languages? *Information Processing Letters*, 28:249–251, 1988.
- [9] L. Fortnow and M. Sipser. Probabilistic computation and linear time. In *Proc. of the 21st ACM Symp. on the Theory of Computing*, pages 148–156, 1989.
- [10] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM J. on Computing*, 18(1):186–208, 1989.
- [11] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 73–90. JAI Press, 1989.
- [12] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, Reading, Mass., 1979.
- [13] O. H. Ibarra. A note concerning nondeterministic tape complexities. *J. ACM*, 19(4):608–612, 1972.
- [14] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. of the ACM*, 39(4):859–868, October 1992.

- [15] W. J. Paul, E. J. Prauß, and R. Reischuk. On alternation. *Acta Informatica*, 14:243–255, 1980.
- [16] W. Ruzzo. On uniform circuit complexity. *J. of Computer and System Sciences*, 22:365–381, 1981.
- [17] A. Shamir. $IP = PSPACE$. *J. of the ACM*, 39(4):869–877, October 1992.
- [18] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. of Computing*, 6:84–85, 1977. See also erratum 7:118, 1978.
- [19] I. Sudborough. Time and tape bounded auxiliary pushdown automata. *Mathematical Foundations of Computer Science*, pages 493–503, 1977.
- [20] I. Sudborough. On the tape complexity of deterministic context free languages. *J. of the ACM*, 25(3):405–414, 1978.