Logspace Verifiers, NC, and NP (Extended Abstract)

Satyanarayana V. Lokam[1], Meena Mahajan[2], V Vinay[3*]

[1] Department of Computer Science, The University of Chicago
Chicago, IL 60637, U. S. A. . E-mail: satya@cs.uchicago.edu
[2] The Institute of Mathematical Sciences, C.I.T. Campus, Madras 600 113, India.
E-mail: meena@imsc.ernet.in
[3] Department of Computer Science and Automation, Indian Institute of Science,
Bangalore 560 012, India. E-mail: vinay@csa.iisc.ernet.in

Abstract. We explore the connection between public-coin interactive proof systems with logspace verifiers and $\mathcal{NC}$ using two different approaches. In the first approach, we describe an interactive proof system for accepting any language in $\mathcal{NC}$ after a logspace reduction, where the verifier is logspace-bounded and the protocol requires polylog time. These results are proved by describing $\mathcal{NC}$ computations as computations over arithmetic circuits using `maximum` and `average` gates, and then translating the arithmetic circuits into interactive proof systems in a natural way. In the second approach, we give a *characterization* of $\mathcal{NC}$ in terms of interactive proof systems where the verifier is logspace-bounded and runs in polylog time. The equivalent interactive proof systems work with error-correcting encodings of inputs, using the polylogarithmically checkable codes introduced in the context of transparent proofs.
We also characterize $\mathcal{NP}$ and $\mathcal{PSPACE}$ via public-coin interactive proof systems where the verifier is logspace-bounded, but has restricted access to auxiliary storage.

1 Introduction

An interactive proof system is a probabilistic game between an infinitely powerful prover and a randomized verifier. This model was introduced in [GMR85], and independently, in [Bab85](see also [BM88]). In [GMR85], the coin tosses of the verifier are hidden from the prover (this is called the *private coins* model), whereas in [Bab85, BM88], the coin tosses of the verifier are visible to the prover (this is called the *public coins* model). It was shown that for polynomial time verifiers both models have essentially the same computational power [GS86]. Generalizing the close connection between alternating Turing machines (ATMs) and interactive proof systems [LFKN92, Sha92, She92], Fortnow and Lund [FL93] give simulations between ATMs with simultaneous space and time bounds and

* Work done while the author was with the Institute of Mathematical Sciences, Madras 600 113, India.

interactive proof systems with space and time bounded verifiers. Condon, Lipton, Ladner [CL88, CL89, Con91, CL92], and Fortnow [For89, Ch.3], examined the power of logspace verifiers under many restrictions. In particular, it was shown [CL92] that with a logspace verifier, public coins can be strictly weaker than private coins, assuming $\mathcal{P} \neq \mathcal{NP}$. In this paper we extend some results of [For89, Ch.3], [FL93], and [CL92].

We study public-coin interactive proof systems with a space-bounded verifier. In Section 2, we define unbounded fanin arithmetic circuits with gates computing `maximum` and `average`. We establish a correspondence between *uniform* families of such circuits and public-coin interactive proof systems. The depth of the circuit equals the number of rounds while the size is exponential in the space of the verifier (Theorem 3). Similar circuits with bounded fanin have earlier been used in [For89, Ch.3], following intuition from [GS86], in the context of logspace verifiers. We use them in a more general setting. In particular, we consider *unbounded fanin* circuits so we can relate depth to rounds; due to non-associativity of the `average` operator, conversion from unbounded fanin to bounded fanin is not clear. We also ensure uniformity of our circuits. Both these differences are crucially used in our proofs to obtain results that properly generalize results from [For89, Ch.3].

In Section 3, we describe how to arithmetize computations of ATMs. A simple low-degree arithmetization, and a size-efficient arithmetization that requires the input to be present in an encoded form, are obtained (Theorems 4 and 5). These results build on techniques developed in [FL93]. Exploiting the fact that the verifier in our case has random access to encoded input, we are able to obtain arithmetizations of significantly smaller size than in [FL93].

In Section 4, we give public-coin interactive proof systems for $\mathcal{NC}$. Using the low-degree arithmetization of Section 3, we first show how to represent ATM computations as computations over the arithmetic circuits described in Section 2. Combining this with the results of Section 2, we get interactive proof systems for accepting any language in $\mathcal{NC}$ after a logspace reduction, where the verifier is logspace bounded and the protocol runs in polylog time (Theorem 6).

It was shown in [FL93] that every language in $\mathcal{NC}$ has a public-coin interactive proof system with a verifier using $O(\log n)$ space and $O(n \log^2 n)$ time. Our above-mentioned result identifies the polynomial-time bottleneck and isolates it in the logspace reduction, so that the rest of the protocol runs fast. Omitting the logspace reduction gives a polylog round interactive proof system for $\mathcal{NC}$.

In Section 5, we characterize $\mathcal{NC}$ exactly via interactive proof systems of polylog time and logspace complexity. Since a sublinear time verifier cannot even read the entire input, we supply the input in an *error-correcting encoding*, using the codes introduced in the context of transparent proofs [BFLS91] (see also [BF93]). Further, the encoding itself is still computable in $\mathcal{NC}$ and is probabilistically checkable in polylog time. Using this encoding, denoted γ, and the size-efficient arithmetizations of Section 3, we show that if a language L is in $\mathcal{NC}$ then $\gamma(L)$ is accepted by interactive proof systems with logspace verifier and polylog running time, where $\gamma(L)$ denotes the set of codewords corresponding to strings in L. Combining this with the reverse simulation from [FL93], we get the claimed equivalence of $\mathcal{NC}$ and interactive proof systems (Theorem 12).

In the last section of this paper, we show an interesting variant of the following result in [CL92]. Let $\mathcal{IPS}(a,b,c)$ denote the class of languages recognized by interactive proof systems in which a specifies public/private coins, b specifies the power of the verifier (polytime, logspace etc.,), and c gives the number of random bits used. In [CL92] the following result is proved: $\mathcal{NP} = \mathcal{IPS}$*(private, logspace, log random bits). On the other hand,* $\mathcal{NLOG} \subseteq \mathcal{IPS}$*(public, logspace, log random bits)* $\subseteq \mathcal{LOGCFL}$.

We show that this weakening of the public coins model is offset by allowing the verifier access to an additional storage in the form of a pushdown stack. Thus, the verifier is a randomized $\mathcal{LOGCFL}$ machine. More precisely, we show (Theorem 13): $\mathcal{NP} = \mathcal{IPS}$*(public, logspace + PDA, log random bits)*. This strengthens another result in [CL92], namely $\mathcal{NP} = \mathcal{IPS}$*(public, polytime, log random bits)*, by replacing the polytime verifier by a $\mathcal{LOGCFL}$ verifier.

We also observe that by relaxing the pushdown stack to a readable stack (push-pop changes are restricted to the top, but stack contents can be read arbitrarily), we can capture $\mathcal{PSPACE}$. In fact, the verifier alternates between pushdown mode and scan mode (read internal contents) only once, and the stack is non-erasing (Theorem 14). Such a verifier is not known to be as powerful as $\mathcal{P}$ [JK89, VC90] and can potentially be weaker. Hence this is a conditional improvement on the standard poly-time verifier for $\mathcal{PSPACE}$ [LFKN92, Sha92, She92].

Notation: In the following, IPRDSP$(r(n), s(n))$ and IPTISP$(t(n), s(n))$ denote the sets of languages accepted by public-coin interactive proof systems using $r(n)$ rounds and $t(n)$ time, respectively, and with the verifier using $s(n)$ space. Also, ATISP$(t(n), s(n))$ denotes the set of languages accepted by alternating Turing machines running in time $t(n)$ and space $s(n)$.

2 Arithmetic circuits and interactive proof systems

In this section, we consider a family of arithmetic circuits defined as follows. The circuit has two types of unbounded fanin gates: MAX and AVG. A MAX gate outputs the maximum of all its input values. An AVG gate returns the arithmetic mean of the values of all its inputs. In addition, there are special input gates called READ gates. These gates have one arithmetic-valued input i, one arithmetic-valued output o, and further one Boolean input b which is a circuit input. If $b = 1$, then $o = i$, otherwise $o = 0$. Thus at a READ gate, a circuit input can filter out an arithmetic value. We make the simplifying assumption that circuit inputs are read only through READ gates. All MAX and AVG gates have as inputs either the outputs of other gates or one of the arithmetic constants 0 or 1. The circuit returns a value at least 3/4 if the input is to be accepted, and a value at most 1/4 otherwise. (We do not consider circuits which return values in the range (1/4,3/4).) Such circuits, of size $S(n)$ and depth $D(n)$, form the family MAX,AVG$(S(n), D(n))$. Unless otherwise stated, we will assume that the circuit families are uniform in the following sense: In a family of circuits of size $S(n)$, a description of the circuit can be generated by a DSPACE($\log S(n)$) machine, given input 1^n.

Apart from having unbounded fanin, our circuits differ from those of [For89, Ch.3] in that we use READ gates to read all circuit inputs. This permits the uniform constructibility of our circuits.

Our main result in this section shows that IPRDSP($r(n), s(n)$) is equivalent, *via* DSPACE($s(n)$)-reductions, to the class of languages accepted by uniform MAX,AVGcircuit families of size $2^{s(n)}$ and depth $r(n)$.

Lemma 1. *IPRDSP*$(r(n), s(n)) \quad \leq_{\mathcal{DSPACE}(s(n))} \quad$ MAX,AVG$(2^{s(n)}, r(n))$.

Proof Sketch: The idea behind converting the interactive proof system to a circuit is as follows. Each configuration of the interactive proof system is a gate of the circuit. In the interactive proof system, when it is the prover's turn to respond, the prover chooses, out of all potential answers, that which maximizes the chances of convincing the verifier. So a prover's move from a configuration is captured by a MAX gate, whose children are configurations resulting from each potential prover response. Similarly, consider verifier's moves. We assume, w.l.o.g., that the verifier makes at most $s(n)$ coin tosses in a round, and they are all made at the beginning of the round. The coin toss sequence is captured by an unbounded fanin AVG gate. Deterministic computation by the verifier, as long as it does not refer to the input, can be "short-circuited" by the uniformity machine; the configuration c before the deterministic computation is a MAX gate with the configuration c' after the deterministic computation as its only input. However, if in the deterministic computation, the verifier accesses input bits, then this short-cutting is not possible, since the uniformity machine does not have access to input bits and so cannot compute c'. This is where the reduction steps in; for each pair of verifier configurations $\langle c, c'\rangle$, it outputs a bit $y_{c,c'}$, which is 1 iff the verifier can, from configuration c and using only deterministic moves (which may depend on the actual input), reach configuration c'. The uniformity machine can now construct a depth-two sub-circuit for the deterministic part of the verifier's computation: guess, through a MAX gate, the configuration c', and verify, by accessing the bit $y_{c,c'}$ through a READ gate, that the configuration is reachable. □

Lemma 2. MAX,AVG$(S(n), D(n)) \subseteq$ *IPRDSP*$(D(n), \log(S(n)))$.

Theorem 3. *IPRDSP*$(r(n), s(n)) \quad \equiv_{\mathcal{DSPACE}(s(n))} \quad$ MAX,AVG$(2^{s(n)}, r(n))$.

3 Arithmetizing the computations of ATMs

In this section we describe arithmetizations of ATISP($t(n), s(n)$) computations. To construct these, we borrow much of the technical machinery developed in [FL93]. However, the size of the arithmetizations in [FL93] is too high to obtain a characterization of $\mathcal{NC}$ (see Section 5). Exploiting the fact that the verifier in our case has random access to encoded input, we can obtain arithmetizations of significantly smaller size. In our proofs, we use the polynomial extension encoding scheme of [BFLS91]. Details of the constructions and proofs will appear in the full version.

Let $L \in \mathrm{ATISP}(t(n), s(n))$ and let M be an ATM accepting L. We assume that M is in a normal form, described in [FL93]. Given an input x, we will construct an arithmetic expression whose value determines whether $x \in L$. Fix a finite field $\mathcal{F}$, a subset $\mathcal{H} \subset \mathcal{F}$ with $h := |\mathcal{H}|$, and a parameter t; the values $|\mathcal{F}|, h$, and, t will be specified later. In the following recursive definition for V_i, when $I_1, \ldots, I_t \in \mathcal{H}$, the t-tuple describes a configuration of M, the function Q defines the validity of M's move from I to I' after existential guess a followed by universal guess b, and R_x describes that M accepts in its final configuration.

$$V_i(I_1, \ldots, I_t, x) = \begin{cases} R_x(I_1, \ldots, I_t) \text{ if } i = 0, \\ \coprod_{a=0}^{1} \prod_{b=0}^{1} \sum_{I' \in \mathcal{H}^t} Q(I_1, \ldots, I_t, I'_1, \ldots, I'_t, a, b) \cdot V_{i-1}(I'_1, \ldots, I'_t, x) \\ \qquad \text{otherwise.} \end{cases} \tag{1}$$

where $\coprod_{x=0}^{1} p(x) := 1 - \prod_{x=0}^{1}(1 - p(x))$. For a fixed x, V_i maps $\mathcal{F}^t$ to $\mathcal{F}$.

Theorem 4. *Let $T := t(n)/2$, and V_i be defined by (1). If $I_{01} \ldots I_{0t} \in \mathcal{H}^t$ is the encoding of the initial configuration of M, then $V_T(I_{01}, \ldots, I_{0t}, x) = 1$ if M accepts x, and, 0 otherwise.*
Furthermore, the degree of each variable in Q and R_x is at most h.

The arithmetizations described above are of low degree but not of small size. Here *size* is the number of internal nodes in the "expression tree" of a formula. The *depth* of a formula is the length of the longest path from the root to a leaf in this tree. The formula size can be reduced if there is random access to encoded input. Accordingly, a different arithmetization is constructed below. The functions F and G_x have essentially the same meanings as Q and R_x above.

$$\Phi_i(I_1, \ldots, I_t, x) = \begin{cases} G_x(I_1, \ldots, I_t) \text{ if } i = 0, \\ \coprod_{a=0}^{1} \prod_{b=0}^{1} \sum_{I' \in \mathcal{H}^t} F(I_1, \ldots, I_t, I'_1, \ldots, I'_t, a, b) \cdot \Phi_{i-1}(I'_1, \ldots, I'_t, x) \\ \qquad \text{otherwise.} \end{cases} \tag{2}$$

Theorem 5. *Let $T := t(n)/2$ and Φ be defined by (2). If $I_{01} \ldots I_{0t} \in \mathcal{H}^t$ is the encoding of the initial configuration of M, then $\Phi_T(I_{01}, \ldots, I_{0t}, x) = 1$ if M accepts x, and, 0 otherwise.*
Furthermore, the degree of any variable in F is at most $O(h)$, $Size(F) = O(h^2 s^2(n))$, and $Depth(F) = O(\log s(n) + \log h)$; also, the degree of each variable in G_x is at most $O(\log n)$, $Size(G_x) = O(h^2 \log^2 n)$, and, $Depth(G_x) = O(\log h + \log\log n)$.

Some explanation as to the difference between (1) and (2) is in order here. In the interactive proof systems based on (1), the function $R_x(I_1, \ldots, I_t)$, for $I \in \mathcal{F}^t$, is evaluated, for a fixed x, by the reduction machine and is used to generate an input to the MAX, AVG circuit that is later simulated by an interactive proof system. On the other hand, in the interactive proof systems based on (2), the verifier is directly given random access to the polynomial extension code $\tilde{x}$ of x. The verifier uses the value of $\tilde{x}$ at a point in $\mathcal{F}^t$ to evaluate G_x. Since we want the verifier to run in polylog time, we need to ensure that the arithmetic expressions are small so they can be evaluated fast.

4 Fast logspace verifiers for $\mathcal{NC}$

In this section, we show a logspace reduction from $\mathcal{NC}$ languages to languages in IPTISP($polylog(n)$, $\log n$). Our main result is:

Theorem 6. $\mathcal{NC} \quad \leq_{\mathcal{LOG}} \quad$ *IPTISP*$(log^{O(1)}(n), \log n)$.

First we observe that directly constructing an interactive proof system to verify the recurrence of Theorem 4 in the standard way, we get a protocol with polylog rounds, provided the parameter t in Theorem 4 is chosen carefully. Thus,

Theorem 7. $\mathcal{NC} \subseteq$ *IPRDSP*$(log^{O(1)}(n), \log n)$

However this construction will not directly give Theorem 6. For this, we first reduce $\mathcal{NC}$ to circuits over MAX and AVG gates. Then we follow the conversion of Lemma 2, and identify points where the reduction can be tightened to make the conversion more time-efficient.

Lemma 8. $\mathcal{NC}^k \quad \leq_{\mathcal{LOG}} \quad$ MAX,AVG$(n^{O(1)}, \log^{k+1} n)$.

Proof Sketch: For simplicity of notation, let $(\mathcal{MAX}, \mathcal{AVG})^k$ denote the class MAX,AVG$(n^{O(1)}, \log^k n)$.

We use the ATM definition of $\mathcal{NC}$; $\mathcal{NC}^k$ = ATISP$((\log n)^k, \log n)$. The ATM computation is arithmetized as in Theorem 4. The $(\mathcal{MAX}, \mathcal{AVG})^{k+1}$ circuit we construct will essentially simulate an LFKN-like protocol on recurrence (1) for verifying that $V_T(I_{01}, \ldots, I_{0t}, x) = 1$. Thus in the $(\mathcal{MAX}, \mathcal{AVG})^{k+1}$ circuit, gate labels have the following information: gate type (MAX or AVG), depth of polynomial (subscript i of formula V_i currently being checked), claimed value of polynomial currently being checked, values of variables which have been initialised, and in the case of an AVG gate, the coefficients of a univariate polynomial of degree h. The root gate of the MAX, AVG circuit has the information: gate type = MAX, depth = T, value = 1, and initialised variables $I_{01}, \ldots, I_{0t}$ where I_0 represents the initial ID of the ATM.

We only mention the points of difference from the standard protocol.

When the equality to be verified involves a computation of the function R, i.e. at a gate labelled $\langle \text{MAX}, 0, v, \alpha_1, \ldots, \alpha_t \rangle$ where the equality to be checked is $v = R(\alpha_1, \ldots, \alpha_t)$, the desired circuit connection is that the gate receives 1 if the equality is satisfied, and 0 otherwise. However to know what constant is to be fed to the gate, the uniformity machine will have to evaluate R, which requires access to the input bits. Since this is not possible, we again exploit the trick used in the proof of Lemma 1; push the difficult part of the computation into the reduction. Specifically, define $B(\alpha_1, \ldots, \alpha_t, v) = 1$ if $v = R(\alpha_1, \ldots, \alpha_t)$ and 0 otherwise. The reduction machine outputs the string of bits $B(\alpha_1, \ldots, \alpha_t, v)$ (one bit for each $\alpha_1, \ldots, \alpha_t, v \in \mathcal{F}$). These bits are the input bits for the $(\mathcal{MAX}, \mathcal{AVG})^{k+1}$ circuit we construct. Now the uniformity machine connects to the MAX gate a READ gate with the constant input 1 and the circuit input $B(\alpha_1, \ldots, \alpha_t, v)$.

We also need to choose the parameters t and $\mathcal{F}$. The polynomials checked at each stage are of degree h, so there is a potential error probability of $h/|\mathcal{F}|$. There are $T(t+2)$ steps at which variables are randomly instantiated. So, to bound the overall error by $1/4$, $|\mathcal{F}| \in O(Tht)$ suffices.

The circuit size itself must be polynomial, so gate labels should be $O(\log n)$ bits wide. The labels require $O((t+h)\log|\mathcal{F}| + \log T)$ bits. Since $T \in O((\log n)^k)$, it can be ignored here. So we require $(t+h)\log|\mathcal{F}| \in O(\log n)$.

The two constraints are satisfied by choosing $t \in O(\frac{\log n}{\log\log n - \log\log\log n})$. This gives $w \in O(\log\log n - \log\log\log n)$ and $h \in O((\frac{\log n}{\log\log n})^{O(1)})$. Thus $|\mathcal{F}|$ is also polylog in n, and the depth of the circuit is $T(t+2) \in O((\log n)^{k+1})$. □

Sketch of Proof of Theorem 6: The circuits constructed in the above proof are uniform in the sense that a description of the circuit can be generated in space logarithmic in its size. There is another notion of uniformity that can be used for circuit classes. A class of circuits is said to be U_{DC}-uniform if the direct connection language L_{DC} of the circuit, comprising of descriptions of individual edges, can be *recognized* deterministically in time logarithmic in the circuit size [Ruz81]. It is known [Ruz81] that U_{DC}-uniform $\mathcal{NC}$ coincides with $\mathcal{NC}$.

The MAX, AVG circuit constructed in Lemma 8 is "almost" U_{DC}-uniform "almost everywhere". Let us qualify this. Most edges can be recognised in time polylog in circuit size, though not logarithmic. Hence "almost" uniform. An exception is recognising edges which involve a computation of the form $v' = v/Q(\alpha_1, \ldots, \alpha_{2t+2})$. Hence "almost everywhere". However, by modifying the proof in certain nontrivial ways (details appear in the full version), we can push the computation of these edges too into the logspace reduction. If this reduction is used in Lemma 8, and then Lemma 2 is applied, then the result follows. □

5 Characterizing $\mathcal{NC}$ by $\mathcal{IPS}$

In this section, we give an exact characterization of $\mathcal{NC}$ in terms of interactive proof systems where the verifier is restricted to run in polylog time and logspace. To allow this situation, we will require that the verifier be given random access to *encoded* input. We justify this requirement by the fact that such an encoding of the input instance can be done in $\mathcal{NC}$. Furthermore, the verifier can probabilistically check in polylog time if what is presented to him is a valid encoding. Such an encoding scheme was utilized in [BFLS91] in the context of *transparent proofs*. This additional feature added to the earlier model of interactive proofs enables us to give an exact characterization of $\mathcal{NC}$. *Notice that the encoded input is provided to the verifier directly and is not part of the proof as in the model of transparent proofs.*

As in the previous section, we view $\mathcal{NC}$ computations as ATM computations. The arithmetization we use is described in Theorem 5. Using standard techniques, this results in an interactive proof system with a verifier of time-space complexity polynomially related to that of the ATM. Our protocol has the same overall structure as that of [FL93] except in places where the input is accessed. Hence, we do not explicitly describe this protocol; the reader is referred to [FL93] for details. We basically perform an LFKN-type protocol on recurrence (2) for Φ_i. Our protocol essentially differs from [FL93] when $i = 0$, the only place where the verifier needs access to the input. Our parameters (degrees of polynomials, size of the field, etc.,) are also slightly different.

In (2), if $i = 0$, then Φ expresses a **Read** move by M: $\Phi(y_1, \ldots, y_t) = G_x(y_1, \ldots, y_t)$. At this point in the protocol, by induction, the verifier has values $\alpha_1, \ldots, \alpha_t, \beta \in \mathcal{F}$, and he needs to check if $\beta = G_x(\alpha_1, \ldots, \alpha_t)$. Here the Verifier uses the arithmetic expression for G_x given by Theorem 5. During its evaluation, the verifier needs the value of the extension $\tilde{x}$ of x at a point $(c_1, \ldots, c_t) \in \mathcal{F}^t$. He gets this value by appealing to the self-correction features of the encoding of x. From [BFLS91, Theorem 4.6], we know that the verifier can get any such value in $polylog(n)$ time per point. This is the key to making our verifier run in $polylog(n)$ time.

An analysis of the protocol (keeping in mind the reduced size of G_x) yields:

Lemma 9. *Let $\epsilon > 0$ be any constant. Then, in the protocol above, the verifier runs in time $(t(n)s(n)\log n)^{O(1/\epsilon)}$ and in space $O(s(n)\log t(n)/\log\log n)$.*

Let $\gamma : \Sigma^* \longrightarrow \Sigma^*$ be the coding function given by [BFLS91, Theorem 4.6]. For a set $S \subseteq \Sigma^*$, let $\gamma(S)$ be the set of codewords corresponding to strings in S. Using Lemma 9 and the properties of γ, we get,

Theorem 10. *If $L \in ATISP(t(n), s(n))$, then $\gamma(L) \in IPTISP((t(n)s(n)\log n)^{O(1/\epsilon)}, s(n)\log t(n)/\log\log n)$.*

Simulation in the other direction is given by the following theorem:

Theorem 11 (FL93). *$IPTISP(t(n), s(n)) \subseteq ATISP(t(n)\log t(n), s(n))$.*

Now, using Ruzzo's characterization [Ruz81] of $\mathcal{NC}$, we get our main theorem for this section:

Theorem 12. *Let L be any language and let γ be as above. Then*
L is in $\mathcal{NC}$ iff $\gamma(L)$ is in $IPTISP(\ polylog(n)\ , \log n)$.

Proof. ¿From Theorem 10, if $L \in \mathcal{NC}$, then $\gamma(L) \in$ IPTISP($polylog(n)$, $\log n$). For the other direction: Given L, we can compute $\gamma(L)$ in $\mathcal{NC}$ since γ is computable in $\mathcal{NC}$. Now if $\gamma(L)$ is in IPTISP($polylog(n)$, $\log n$), then by Theorem 11, there is an ATM that recognizes $\gamma(L)$ and runs in time $polylog(n)$ and space $O(\log n)$, giving uniform $\mathcal{NC}$ circuits that recognize $\gamma(L)$. Combining these with the $\mathcal{NC}$ circuits that compute γ, we then get $\mathcal{NC}$ circuits to recognize L. $\square$

6 AuxPDA logspace verifiers capture $\mathcal{NP}$

In this section, we give a new characterisation of $\mathcal{NP}$, in terms of an interactive proof system where the verifier is a $\mathcal{LOGCFL}$ machine with a restricted number of coin tosses. This strengthens some results of [CL92].

Theorem 13. *$\mathcal{IPS}$(*public, logspace + PDA, log random bits*) = $\mathcal{NP}$*

Proof Sketch: $\supseteq$: This proof uses ideas similar to [CL92], except that the privacy of coin tosses is replaced by access to a pushdown store. An $\mathcal{NP}$ membership proof can be written as an array, and verification of the proof involves some local row-wise operations and some local column-wise operations. The prover sends the array twice, once in row-major order and once in column-major order. As the array is being sent, the verifier can do the local checks. But now the verifier must also check that the prover did not send two different arrays in the two phases. Here Lipton's fingerprinting technique [Lip90] is used; for each array, a small fingerprint which fits in log space is probabilistically generated. If the arrays are different, the fingerprints differ with high probability. For this to work, the coin tosses involved in fingerprinting the first array should not be known to the prover while he is sending the second array. So the verifier stacks the arrays on the pushdown, and does the fingerprinting while popping the stack. $\square$

The crucial idea used in the above proof is that parts of the verification can be deferred, and the data required for the deferred verification can be stored on the auxiliary storage. In the above case, deferring helped make public coins effectively private. It can also be used to bring down a polynomial time verification to a logspace verification, provided data is stacked in appropriate form.

In particular, considering the verifier in interactive proof system for $\mathcal{PSPACE}$ [LFKN92, Sha92, She92], we get the following result. The stack used by the verifier here is not strictly a pushdown; PUSH and POP are allowed only at top-of-stack, but internal stack contents can be read or scanned.

Theorem 14. *$\mathcal{PSPACE}$ is exactly equal to the class of languages accepted by interactive proof systems where the verifier uses* $\log n$ *space and has access to a non-erasing stack. (Non-erasing means that there are no POP moves.) Further, the class of languages remains unchanged even if the verifier is restricted to change between pushdown mode and scan mode, on the stack, only once.*

Acknowledgments

We are grateful to Jaikumar Radhakrishnan from the Tata Institute of Fundamental Research, Bombay, and to Laci Babai, Lance Fortnow, Ashish Naik, and Janos Simon from the University of Chicago for helpful comments on earlier versions of this work.

References

[Bab85] L. Babai. Trading group theory for randomness. In *Proc. 17th STOC*, pp 421–429, 1985.

[BF93] L. Babai and K. Friedl. On slightly superlinear transparent proofs. Technical Report CS-93-13, Department of Computer Science, University of Chicago, 1993.

[BFLS91] L. Babai, L. Fortnow, L.A. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proc. 23rd STOC*, pp 21–31, 1991.

[BM88] L. Babai and S. Moran. Arthur-Merlin games: a randomized proof system and a hierarchy of complexity classes. *J. Comp. Syst. Sci.*, 36:254–276, 1988.

[CL88] A. Condon and R. Ladner. Probabilistic game automata. *J. Comp. Syst. Sci.*, 36(3):452–489, 1988.

[CL89] A. Condon and R. Lipton. On the complexity of space-bounded interactive proofs. In *Proc. 30th FOCS*, pp 462–467, 1989.

[CL92] A. Condon and R. Ladner. Interactive proof systems with polynomially bounded strategies. In *Proc. 7th Conference on Structure in Complexity Theory*, pp 282–294, 1992.

[Con91] A. Condon. The complexity of the max word problem and the power of one-way interactive proof systems. In *Proc. 8th STACS*, pp 456–465, 1991. LNCS 480.

[FL93] L. Fortnow and C. Lund. Interactive proof systems and alternating time-space complexity. *Theoretical Computer Science*, 113:55–73, 1993. also in Proc. 8th STACS 1991, LNCS 480.

[For89] L. Fortnow. *Complexity-theoretic aspects of interactive proof systems*. PhD thesis, M. I. T., May 1989. Tech. Rep. MIT/LCS/TR-447.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proc. 17th STOC*, pp 291–304, 1985. full version in SIAM J. Comput., Vol 18(1), pp 186–208.

[GS86] S. Goldwasser and M. Sipser. Private coins versus public coins in interactive proof systems. In *Proc. 18th STOC*, 1986. also in *Advances in Computing Research 5: Randomness and Computation*, JAI Press, Greenwich, CT, 1989.

[JK89] B. Jenner and B. Kersig. Characterizing the polynomial hierarchy by alternating auxiliary pushdown automata. *RAIRO Theoretical Informatics and Applications*, 23:93–99, 1989. also in Proc. STACS(1988), LNCS **294** 118–125.

[LFKN92] C. Lund, L. Fortnow, H. Karloff, and N. Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, October 1992. also in Proc. 31st FOCS 1990, pp 1–10.

[Lip90] R. J. Lipton. Efficient checking of computations. In *Proc. 7th STACS*, pp 207–215, 1990. LNCS 415.

[Ruz81] W.L. Ruzzo. On uniform circuit complexity. *J. Comput. Syst. Sci.*, 22:365–383, 1981.

[Sha92] A. Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992. also in Proc. 31st FOCS 1990, pp 11–15.

[She92] A. Shen. IP = PSPACE: Simplified proof. *J. ACM*, 39(4):878–880, 1992.

[VC90] V. Vinay and V. Chandru. The expressibility of nondeterministic auxiliary stack automata and its relation to treesize bounded alternating auxiliary pushdown automata. In *Proc. 10th FST & TCS*, pp 104–114, 1990. LNCS 472.