Computationally Limited Randomness *

Matei David[1] Phuong Nguyen[2] Periklis A. Papakonstantinou[3] Anastasios Sidiropoulos[4]

[1]Center for Computational Intractability, Princeton University, Princeton, NJ, USA

[2]Department of Computer Science, McGill University, Montreal, QC, Canada

[3]ITCS, Tsinghua University, Beijing, PR China

[4]Toyota Technological Institute, Chicago, IL, USA

mateid@cs.princeton.edu pnguyen@cs.toronto.edu papakons@tsinghua.edu.cn tasos@ttic.edu

Abstract: The starting point of this work is the basic question of whether there exists a formal and meaningful way to limit the *computational power* that a time bounded randomized Turing Machine can employ *on its randomness*.

We attack this question using a fascinating connection between space and time bounded machines given by Cook [Coo71]: a Turing Machine $\mathcal{S}$ running in *space* s with access to an *unbounded stack* is *equivalent* to a Turing Machine $\mathcal{T}$ running in *time* $2^{O(s)}$. We extend $\mathcal{S}$ with access to a read-only tape containing $2^{O(s)}$ uniform random bits, and a usual error regime: one-sided or two-sided, and bounded or unbounded. We study the effect of placing a bound p on the number of passes $\mathcal{S}$ is allowed on its random tape. It follows from Cook's results that:

- If $p = 1$ (one-way access) and the error is *one-sided unbounded*, $\mathcal{S}$ is equivalent to *deterministic* $\mathcal{T}$.
- If $p = \infty$ (unrestricted access), $\mathcal{S}$ is equivalent to *randomized* $\mathcal{T}$ (with the same error).

As our first two contributions, we completely resolve the case of unbounded error. We show that we cannot meaningfully interpolate between deterministic and randomized $\mathcal{T}$ by increasing p:

- If $p = 1$ and the error is *two-sided unbounded*, $\mathcal{S}$ is still equivalent to *deterministic* $\mathcal{T}$.
- If $p = 2$ and the error is *unbounded*, $\mathcal{S}$ is already equivalent to *randomized* $\mathcal{T}$ (with the same error).

In the bounded error case, we consider a logarithmic space Stack Machine $\mathcal{S}$ that is allowed p passes over its randomness. Of particular interest is the case $p = 2^{(\log n)^i}$, where n is the input length, and i is a positive integer. Intuitively, we show that $\mathcal{S}$ performs polynomial time computation on its input and parallel (preprocessing plus NC^i) computation on its randomness.

Formally, we introduce *Randomness Compilers*. In this model, a polynomial time Turing Machine gets an input x and outputs a (polynomial size, bounded fan-in) circuit C_x that takes random inputs. Acceptance of x is determined by the acceptance probability of C_x. We say that the randomness compiler has depth d if C_x has depth $d(|x|)$. As our third contribution, we show that:

- $\mathcal{S}$ simulates, and is in turn simulated by, a randomness compiler with depth $O\left((\log n)^i\right)$, and $O\left((\log n)^{i+1}\right)$, respectively.

Randomness Compilers are a formal refinement of polynomial time randomized Turing Machines that might elicit independent interest.

Keywords: randomness; limited randomness; probabilistic polynomial time; hierarchy; stack machine

*M.D. is supported in part by NSF grant CCF-0832797, P.N. is funded by an NSERC postdoctoral fellowship, P.P. is supported in part by the National NSF China Grant 60553001, 61073174, 61033001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

1 Introduction

Most deterministic computational devices can be extended by providing them access to a uniform random string, and allowing them to err on every input with some probability. Several error regimes are usually investigated. We say that the error is *bounded* if it can be reduced, e.g., to an arbitrary constant, and *unbounded* otherwise. In the case of decision problems, to which we restrict our attention, we say that the error is *one-sided* if the device makes only false negatives error, and *two-sided* otherwise. Randomization with one-sided unbounded error is commonly referred to as *nondeterminism*. Randomness might or might not add power to the underlying computational device. For example, randomness *does not* add any power to an unrestricted Turing Machine, but it *does* add power to an efficient two-party communication protocol, see e.g. [KN97]. Understanding the effect of randomness on the power of a time or space bounded Turing Machine is the subject of some of the most fundamental open problems in theoretical computer science; see e.g. [KI04] for results and references within, and e.g. [Nis94].

Our motivation in this paper is the question of whether there exists a formal and meaningful way to limit the computational power that a time bounded randomized Turing Machine can employ towards its randomness, all while keeping its original power towards its input. We attack this question by using an equivalence between time and space bounded computation given by Cook [Coo71]. Our approach is further motivated by some fundamental questions involving the effect of randomness on the power of space bounded Turing Machines.

Several methods have been considered to give a Turing Machine access to randomness. A common way is to implement the access to randomness inside the transition function. Alternatively, this can be seen as having access to an one-way tape that contains a uniform random string. Henceforth, we refer to this as *read-once* or *one-way access to randomness*. Most randomized computational complexity classes are defined in terms of read-once randomness. Another way, referred to as *two-way access to randomness* or *tape randomness*, is to provide the machine with an auxiliary read-only tape containing uniform random bits. In this case, the machine can control the movement of the input head on the extra tape, using the random bits only when needed. Also, note that in this case the length of the random tape may significantly affect the power of the machine. Finally, a third way is to append randomness as part of the input, a common way to add randomization to combinatorial circuits. Of these three methods, tape randomness is the most general since it naturally simulates the others. Tape randomness, however, allows for more refinements. For example, much effort in the areas of derandomization and pseudorandomness has been placed on the task of reducing the *amount of randomness* used by a Turing Machine. In this case, the context of the random string is replaced by a pseudo-random one of much smaller entropy.

Randomness in Space Bounded Computation

Traditionally, space bounded complexity classes such as RL and NL are defined using read-once randomness. The need to consider (potentially more powerful) two-way access to randomness becomes apparent e.g. when space bounded Turing Machines are used to simulate other computational models. Consider, for instance, the known fact (e.g. [DK00]) that a logarithmic space Turing Machine simulates a polynomial size, logarithmic depth, bounded fan-in circuit (i.e., $\mathsf{NC}^1 \subseteq \mathsf{L}$). We emphasize that a similar relation between the respective randomized extensions (i.e., $\mathsf{RNC}^1 \subseteq \mathsf{RL}$) does *not* follow immediately, and in fact it is *not known to hold*. The reason is that in a randomized circuit the randomness is appended to the input, so the circuit can inspect both with common resource bounds. On the other hand, it is easy to see that this model is simulated by a logarithmic space Turing Machine with *unrestricted (two-way) tape randomness*.

In this paper, we investigate the difference between *read-once* and *general tape randomness* in a variant of space bounded Turing Machines. Before proceeding any further, it is important to point out that this is a non-issue in the case of *time-bounded* Turing Machines(!) If such a machine is provided with either read-once or tape randomness, then it can simply save the random bits it uses on a separate work tape, and subsequently retrieve any of them. However, this scheme no longer works in the case of a *space-bounded* Turing Machine, and the results in [KV85, Nis93], and in part in this work, suggest that its power is indeed greatly influenced by its capacity to *recall and reuse random bits*.

Stack Machines

A stack naturally models an unbounded storage space that comes equipped with a first-in last-out access restriction. While ubiquitous in algorithm design, the stack has found significant applications in complexity theory, where it was used to prove connections between several important models of computation. First, note that it is not useful to add a stack to a Turing Machine that has no restrictive space bound (e.g., a purely time-bounded machine), because a stack can easily be simulated by an additional work tape. Second, note that a space bounded Turing Machine with two unbounded stacks and no additional work space can simulate an unrestricted Turing Machine (by "juggling" the input between the two stacks). There are, however, highly nontrivial consequences to adding *a single, unbounded stack* to a *space-bounded Turing Machine*. This model was previously referred to as an "auxiliary push-down automaton" (AuxPDA). In this work, we prefer the simpler term *Stack Machine*.

There are no known equivalences between purely space and time bounded Turing Machines. The main premise of this work is a result of Cook [Coo71], who showed that such a connection does exist when the space bounded machine is augmented with a stack.

Theorem 1 ([Coo71])**.** *A deterministic Stack Machine $\mathcal{S}$ running in* space s *is equivalent to a deterministic Turing Machine $\mathcal{T}$ running in* time $2^{O(s)}$.

We extend $\mathcal{S}$ with access to a read-only tape containing $2^{O(s)}$ uniform random bits, on which $\mathcal{S}$ is allowed p passes[1]. We consider the usual error conditions: one-sided or two-sided, and bounded or unbounded. It follows from Cook's results [Coo71] that:

Corollary 2 ([Coo71])**.**

- *If $p = 1$ (one-way access) and the error is* one-sided unbounded*, $\mathcal{S}$ is equivalent to* deterministic $\mathcal{T}$.
- *If $p = \infty$ (unrestricted access), $\mathcal{S}$ is equivalent to* randomized $\mathcal{T}$ *(with the same error).*

Moreover, although there are no known (or believed) equivalences between simultaneous time-space bounded Turing Machines and size-depth bounded families of circuits, these equivalences hold if instead of Turing Machines we have Stack Machines; e.g. [BCD$^+$89, Ruz80, Ruz81, Ven91].

1.1 Our Results

There is a large gap between allowing the Stack Machine $\mathcal{S}$ one-way and unrestricted access to the random tape. We study the effect of increasing the number of passes p that $\mathcal{S}$ is allowed on its random tape from 1 (corresponding to one-way access) to ∞ (corresponding to unrestricted access). As suggested by Corollary 2 (for one-sided error), this increases the power of $\mathcal{S}$ from deterministic to randomized $\mathcal{T}$.

Unbounded Error

As our first contribution, we extend Cook's result on the power of read-once access to tape randomness, showing that even if the error is *two-sided unbounded*, $\mathcal{S}$ still does not gain any additional power.

Theorem 3. *If $p = 1$ (one-way access) and the error is* two-sided unbounded*, $\mathcal{S}$ is still equivalent to* deterministic $\mathcal{T}$.

Informally, we achieve this by showing how to *compute exactly* in time $2^{O(s)}$, for every pair of "surface configurations" of the Stack Machine, the probability that if we start in one we reach the other at the same stack level, without having popped the initial top stack symbol. For comparison, for Cook's one-sided unbounded error result, one only needs to compute whether the corresponding probabilities are positive.

As our second contribution, we completely resolve the case of unbounded error. We show that, perhaps surprisingly, allowing *as little as 2 passes* over the random tape is equivalent to having *no pass bound at all.* Thus, in the case of unbounded error, there is no meaningful way to interpolate between the power of the deterministic and randomized versions of the time bounded Turing Machine $\mathcal{T}$.

Theorem 4. *If $p = 2$ (two passes) and the error is* unbounded*, $\mathcal{S}$ is already equivalent to* randomized $\mathcal{T}$ *(with the same error).*

[1] We assume the passes alternate in direction, so that p passes are equivalent to $p - 1$ reversals.

Informally, to achieve this result we prove that every path in the computation tree of the randomized Turing Machine can be encoded as a "special" certificate that can be checked by the Stack Machine. Most certificates of the Stack Machine are not special, but this is inconsequential in the case of unbounded error.

Bounded Error

In the bounded error case, we restrict our attention to comparing a *logarithmic space* Stack Machine $\mathcal{S}$ with p passes over a *polynomially long* random tape with a *polynomial time* Turing Machine $\mathcal{T}$ (i.e., we set $s = \log n$ in Corollary 2). By Corollary 2 and Theorem 3, we know that by increasing p from 1 to ∞, $\mathcal{S}$ ranges in power from deterministic to randomized $\mathcal{T}$. Naturally, this hierarchy collapses to level $p = 1$ if one were to show, e.g., that $\mathsf{P} = \mathsf{BPP}$. Moreover, the collapse demonstrated in Theorem 4 (level $p = 2$ equals level $p = \infty$) crucially hinges on the ability of the Stack Machine to decide based only on a minimal advantage drawn from exponentially few random strings. Informally, it seems that this can only be as easily achieved in a regime of unbounded error.

In light of the discussion above, we find it interesting to ask what kind of computational power is achieved at intermediate levels of the bounded-error hierarchy. In particular, it is natural to ask whether there exists some *refinement of the randomness in the time bounded Turing Machine* $\mathcal{T}$ that is captured by restricting the number of passes over tape randomness in the Stack Machine $\mathcal{S}$. In answering this question, we are inspired by the line of work in [Ruz81, Ruz80, BCD$^+$89], leading to the result of Allender [All89].

Theorem 5 ([All89]). *A logarithmic space Stack Machine that makes* $2^{O\left((\log n)^i\right)}$ input head moves *is roughly equivalent* [2] *to a polynomial time uniform circuit with bounded fan-in gates, polynomial* size*, and* depth $O\left((\log n)^i\right)$ *(i.e.,* NC^i*).*

Returning to the Stack Machine $\mathcal{S}$ which has unrestricted access to the input and $p = 2^{O\left((\log n)^i\right)}$ passes over the random tape, we observe that, intuitively, $\mathcal{S}$ seems to perform two types of computation: an arbitrary polynomial time computation on its input, and a parallel (NC^i) computation on its random tape. Below, we formalize this intuition and we show that, perhaps surprisingly, these two computations can be formally separated in the following two-phase model.

Consider the following model[3] of computation. A *randomness compiler* R consists of a polynomial time Turing Machine (transducer) M, and it operates as follows. When R is given input x and randomness r:

Phase one: M is given x (alone) and it produces a circuit C_x (obviously, of polynomial size);
Phase two: C_x is given input r;
The output of the compiler R is defined to be the output of C_x on r.

Acceptance/rejection of an input x is defined in the usual way depending on the error condition. We say that a randomness compiler has *depth* d if the depth of the intermediate circuit C_x is at most $d(|x|)$. Intuitively, this is a formal model of computation which separates the power used to access the input and the power used to access the randomness. In particular, this model can simulate both a deterministic polynomial time Turing Machine (by producing a constant circuit), and a randomized polynomial size depth d circuit (when the precomputation is used to select a circuit of the appropriate size from the family, and the input x is substituted inside).

As our third contribution, we show that the power of logarithmic space Stack Machines with pass bounded tape randomness is closely connected to the power of depth bounded randomness compilers.

Theorem 6. *For every positive integer* i*, the following holds. A logarithmic space Stack Machine with* $p = 2^{O\left((\log n)^i\right)}$ *passes over a polynomially long random tape simulates, and is in turn simulated by, a randomness compiler of depth* $O\left((\log n)^i\right)$*, and* $O\left((\log n)^{i+1}\right)$*, respectively.*

Informally, one simulation follows from the known results connecting Stack Machines and circuits. For the nontrivial one (the compiler simulating the randomized Stack Machine), we prove a more technical Time Compression Lemma (not to

[2] We defer a formal statement to Section 1.2

[3] The model of Randomness Compilers was suggested by Mark Braverman.

be confused with other uses of the term in computational complexity), and use it together with appropriately adjusted older works on Stack Machines.

1.2 Related Work

The difference between read-once and unrestricted tape randomness in space bounded Turing Machines has been studied before in the absence of a stack.

A corollary of the work by Karpinksi and Verbeek [KV85] is that a logarithmic space Turing Machine with two-way access to a $2^{n^{O(1)}}$-long random string characterizes PSPACE with zero error. The result of [KV85] suggests that unrestricted tape randomness adds significant power to a space bounded machine.

Nisan [Nis93], shows that when the random tape is of polynomial size, a logarithmic space Turing Machine with read-once randomness and two-sided bounded error can be simulated by a logarithmic space Turing Machine with unrestricted tape randomness and *zero error*, i.e., the machine produces an answer with constant probability, and when it does, it is always correct.

A good reference on the power of randomized space bounded Turing Machines without a stack, is the survey by Saks [Sak96].

Relations and equivalences between Stack Machines and other models of computation (e.g. simultaneous size-depth bounded circuits) have been studied in a long line of work, see e.g. [All89, BCD$^+$89, Coo71, Ruz81, Ruz80, Ven91].

2 Preliminaries

2.1 Notation and conventions

We denote by n the input length. Whenever we use $s = s(n)$ to denote the space bound in a Turing Machine, we assume that $s(n) = \Omega(\log n)$.

We use standard definitions for Turing Machines, circuits, and complexity classes such as $\mathsf{Time}(t)$, $\mathsf{Space}(s)$, L, P, NC^i, AC^i, SAC^i. We use the standard prefixes R-, BP-, N-, P-, to denote one-sided bounded, two-sided bounded, one-sided unbounded, and two-sided unbounded error conditions, respectively. For all randomized Turing Machines we consider, we require that they respect their (time, space) bounds in the worst case with respect to the random string. We use standard notions of uniformity for circuit families. For reference, see e.g., [AB09, DK00].

2.2 Stack Machines

A (decider) Stack Machine is a space bounded Turing Machine with access to an unbounded stack (for more formal definition of the model and the computation, see [Coo71].) Concretely, a Stack Machine with space bound $s = s(n)$ consists of: a finite state control; a read-only input tape of length n; one (or several) read-write work tape(s) of total size $s(n)$; and an unbounded stack. A *(full) configuration* of a Stack Machine consists of: the state ($O(1)$ bits); the position of the head on the input tape ($O(\log n)$ bits); the content and head positions of the work tapes ($O(s)$ bits); and the content of the stack. A *surface configuration* is similar to a full configuration, but only includes the top stack symbol instead of the entire content of the stack. The following are easy to prove:

Fact 7. *Let S be a* decider *Stack Machine with space s. Then, the stack height achieved by S is at most $2^{O(s)}$, and the running time of S is at most $2^{2^{O(s)}}$.*

We add the term -Pd- (for "push-down") to denote the fact that the Turing Machines used to characterize a certain complexity class are augmented with an unbounded stack. Thus, we write $\mathsf{PdSpace}(s)$, PdL, and $\mathsf{PdSpaceTime}(s,t)$ for the classes of languages decided by Stack Machines with space s, logarithmic space, and simultaneous space s and time t, respectively. With this notation, Cook [Coo71] shows that:

Theorem ([Coo71]). $\mathsf{PdSpace}(O(s)) = \mathsf{Time}\left(2^{O(s)}\right)$

For example, $O(\log n)$ space Stack Machines compute exactly the problems in P. Note that, in general, such machines take $2^{n^{O(1)}}$ steps. In fact, one can show that they must take $2^{(\log n)^{\omega(1)}}$ steps, unless $\mathsf{P} = \mathsf{NC}$, e.g. [BCD$^+$89]. Furthermore, logarithmic space Stack Machines running in quasi-polynomial time characterize the NC-hierarchy.

Theorem ([Ruz81]). *For every positive integer i,* $\mathsf{NC}^i \subseteq \mathsf{PdSpaceTime}\left(O(\log n), 2^{O\left((\log n)^i\right)}\right) \subseteq \mathsf{NC}^{i+1}$.

2.3 Randomized Stack Machines

We extend Stack Machines with randomness as follows.

Definition 8. A *randomized Stack Machine* with space s and pass bound $p = p(n)$ is a regular Stack Machine extended with access to a read-only tape of length $2^{O(s)}$ containing uniform random bits, on which the machine is allowed p passes.

We emphasize the fact that there is no bound on the number of passes the Stack Machine is allowed on any of its other tapes (input, work, or stack). We write $p = \infty$ if there is no bound on the number of passes over the random tape.

Remark 9. In our definition, we use a bound on the *length of the random tape* that needs to be justified. When a space s Stack Machine is extended with randomness by allowing it to "flip coins" inside the transition function, it can potentially use as many bits of randomness as its running time. By Fact 7, this can be up to $2^{2^{O(s)}}$ (!) However, we would like a space s Stack Machine with unrestricted (two-way) access to the random tape to be equivalent with a randomized time $2^{O(s)}$ Turing Machine. We stress that such an equivalence is only known to hold when the length of the random tape is at most $2^{O(s)}$.

For every error condition X- (R-, BP-, N-, or P-) we denote by $\mathsf{XPdSpacePasses}\,(s, p)$ the class of problems decided by randomized Stack Machines with space s and p passes over the random tape. We write $\mathsf{XPdL}\,[p]$ for $\mathsf{XPdSpacePasses}\,(O\,(\log n), p)$. With this notation, it follows from Cook [Coo71] that:

Theorem ([Coo71])**.** $\mathsf{NPdSpacePasses}\,(O\,(s), 1) = \mathsf{PdSpace}\,(O\,(s)) = \mathsf{Time}\,(2^{O(s)})$. *Furthermore, for every error condition* X-*,* $\mathsf{XPdSpacePasses}\,(O\,(s), \infty) = \mathsf{XTime}\,(2^{O(s)})$.

3 Unbounded Error

Consider a randomized Stack Machine S with space $O\,(s)$ and p passes over the random tape, as in Definition 8. By the results of Cook [Coo71], we know that if S makes *1 pass* over the random tape ($p = 1$) and *one-sided unbounded* error, S is equivalent with a deterministic Turing Machine with time $2^{O(s)}$. Cook's results do not say anything about the power of S if it is allowed *1 pass* and (potentially more powerful) *two-sided unbounded* error. At the other extreme, we know that when S is allowed *unrestricted access* to the random tape ($p = \infty$), S is equivalent with a *randomized* Turing Machine with time $2^{O(s)}$ and the same error condition.

Our first two contributions completely settle the case of unbounded error. Our results are, in a sense, negative. We show that we cannot meaningfully interpolate between the powers of a deterministic and a randomized Turing Machine with time $2^{O(s)}$ by increasing the number of passes p that a randomized space s Stack Machine is allowed on its random tape. In other words, in this case the hierarchy of classes obtained by increasing the number of passes p is completely degenerate:

- If S is allowed a single pass over its random tape ($p = 1$), it is equivalent to a *deterministic* Turing Machine with time $2^{O(s)}$.
- If S is allowed 2 passes over its random tape ($p = 2$), it is equivalent to a *randomized* Turing Machine with time $2^{O(s)}$.

We achieve this in two steps. First, we show that one pass over the random tape is useless, even if the Stack Machine is allowed two-sided unbounded error.

Theorem 3. $\mathsf{PPdSpacePasses}\,(O\,(s), 1) = \mathsf{Time}\,(2^{O(s)})$

The easy inclusion is

$$\begin{aligned}\mathsf{Time}\left(2^{O(s)}\right) &\subseteq \mathsf{NPdSpacePasses}\,(O\,(s), 1)\\ &\subseteq \mathsf{PPdSpacePasses}\,(O\,(s), 1)\end{aligned}$$

The other direction is a non-trivial extension of Cook's arguments [Coo71]. We need to show that a randomized Stack Machine S with space $O\,(s)$, $p = 1$ pass over the random tape, and two-sided unbounded error, can be simulated by a deterministic Turing Machine T in time $2^{O(s)}$. Intuitively, T must *compute exactly*, for every input x, the probability that S accepts x. To do that, we start with the same notion of "realizable pair" of surface configurations used by Cook: these are two surface configurations C_1, C_2 such that there exist some computation path of S leading from C_1

to C_2 in such a way that the top stack symbol is the same in both, and this symbol is not popped in between them. Cook shows that all realizable pairs can be computed efficiently, which is sufficient when S has *one-sided* unbounded error. In our case, we need to compute, for every such pair, the exact probability (over the choice of the random string) that S will reach C_2 when started in C_1. The full proof is deferred to Appendix B.

An important remark on Theorem 3 is that the proof crucially relies on the fact that the random tape has length at most $2^{O(s)}$. For comparison, Cook's proof for the one-sided unbounded error case does not need this bound. Also, recall Remark 9.

Our second result in the unbounded error case is a strong collapse.

Theorem 4. *For* X- *being either* N- *or* P-, $\mathsf{XPdSpacePasses}\left(O\left(s\right), 2\right) = \mathsf{XPdSpacePasses}\left(O\left(s\right), \infty\right) = \mathsf{XTime}\left(2^{O(s)}\right)$

It is sufficient to show that $\mathsf{XPdSpacePasses}\left(O\left(s\right), 2\right) = \mathsf{XTime}\left(2^{O(s)}\right)$. The $\subseteq$ direction is trivial. The proof of the other simulation, though technically easy, illustrates an interesting and fundamental interaction between the stack and the unbounded error condition. Informally, using the stack and only 2 passes over the random tape, the Stack Machine can check that the certificate on the random tape is of a very particular form; a form that encodes a unique computation path of the randomized Turing Machine.

Proof Sketch of Theorem 4. Let T be a randomized Turing Machine running in time $2^{c_1 \cdot s}$, for some constant c_1. Assume T has one-sided unbounded (nondeterministic) error. (The case of two-sided unbounded error is similar). Our goal is to construct a Stack Machine S with space $O\left(s\right)$ and 2 passes over the random tape that accepts its input x if and only if T accepts x along *some* computation path.

Note that a full configuration of T can be encoded on $2^{c_2 \cdot s}$ bits, for some constant c_2. S uses a random tape of length $2^{(c_1+c_2)\cdot s} = 2^{O(s)}$, which it conceptually divides into $2^{c_1 \cdot s}$ regions $R_1, R_2, \ldots$, each one of them of length exactly $2^{c_2 \cdot s}$. In the first pass over the random tape, S simultaneously checks that: (i) R_i encodes a configuration C_i of T, for odd i; (ii) the *reverse* of R_i encodes a configuration C_i of T, for even i; and (iii) C_i can be followed by C_{i+1} in the computation of T, for odd i. In the second pass, S checks that: (iv) C_i can be followed by C_{i+1} in the computation of T, for even i. Clearly, (i) and (ii) do not require the use of the stack, and $O\left(s\right)$ space is sufficient. Furthermore, it is not much harder to see that (iii) and (iv) can be achieved by pushing C_i on the stack and popping it while scanning C_{i+1}. Finally, S accepts if the last configuration of T is accepting. Then, if T accepts along some computation path π, S accepts with the certificate that encodes the sequence of configurations of T in π. □

4 Randomness Compilers

In this section, we consider the case of randomized Stack Machines with bounded error conditions. We restrict our attention to the case of logarithmic space ($s = O\left(\log n\right)$). We discuss the two-sided bounded error (BP-), but everything applies to one-sided bounded error (R-) as well.

By the results of Cook [Coo71] and Theorem 3, we know that a logarithmic space randomized Stack Machine S with an increasing number p of passes over a polynomially long random tape interpolates between deterministic and randomized polynomial time computation. Two observations are in order. First, if one were to prove such a strong statement as $\mathsf{P} = \mathsf{BPP}$, the entire hierarchy would collapse to level $p = 1$ (or, even $p = 0$). This seems to be a difficult task [KI04], and is not suggested in any way by our results. Second, the collapse shown in Theorem 4 of the associated unbounded error hierarchy crucially depends on the ability of the randomized Stack Machine (that is simulating a randomized Turing Machine) to accept with exponentially small advantage over a random guess, derived from very few "special" certificates. Informally, this seems more like a "glitch" in definitions than evidence that such a collapse can be achieved in the bounded error case. Thus, we find it interesting to ask what kind of computational power is achieved along this hierarchy, and, in particular, whether this power captures a *natural refinement of randomness in polynomial time computation.* In this section, we provide some indication that this is indeed the case, by providing an alternative characterization of the middle layers in terms of other familiar computational devices. In our study, we are inspired by the line of work in [Ruz81, Ruz80, BCD$^+$89, All89] connecting logarithmic space Stack Machines in the presence of other bounds (time or input head moves) with combinatorial circuits.

Consider a logarithmic space randomized Stack Machine S with $p = 2^{O\left((\log n)^i\right)}$ passes over the random tape. Intuitively, S is free to perform arbitrary polynomial time computation with respect to its input [Coo71], but only limited computation with respect to its random tape. In particular, we see that S can perform $2^{O\left((\log n)^i\right)}$ random accesses on the random tape [4], since it is allowed to perform $2^{O\left((\log n)^i\right)}$ passes and the random tape is just polynomially long. Thus, S could easily simulate (using depth first search) the evaluation of a bounded fan-in circuit of depth $O\left((\log n)^i\right)$ whose input gates are the bits on the random tape. Moreover, at every node in this circuit, S could pause the depth first search and perform arbitrary polynomial time computation with respect to its input. Motivated by the discussion above, we define the following model of computation.

Definition 10. A *Randomness Compiler* R consists of a deterministic polynomial time Turing Machine (transducer) M, and it operates as follows. The compiler R is a randomized computational device that takes an input x and a uniform random string r. In the first phase, M is given x, and it outputs a circuit C_x. In the second phase, C_x is given input r. The output of the compiler R is defined to be the output of C_x on r. We say that R computes a language L with two-sided bounded error if it accepts an $x \in L$ and it rejects an $x \notin L$ with error probability $\leq 1/3$.

We say that the compiler R has depth $d = d(n)$ if for every x, C_x has depth at most $d(|x|)$. We denote by $\mathsf{P{+}BPNC}^i$ the class of languages accepted with one-sided bounded error by Randomness Compilers with depth $O(\log^i n)$.

Remark 11. To clarify this definition, recall the definition of P-uniform BPNC^i. A language L is computed by a P-uniform BPNC^i circuit family if the following holds. On input (x, r), where $|r| \leq |x|^{O(1)}$, a polynomial time Turing Machine is given input 1^n, and it outputs a circuit C_m, where $m = |(x, r)|$. This circuit has size $n^{O(1)}$, depth $O\left((\log n)^i\right)$, and gates with bounded fan-in. The circuit C_m is then given input (x, r). We are guaranteed that for every $x \in L$, (x, r) is accepted with probability at least $2/3$, and for every $x \notin L$, (x, r) is rejected with probability at least $2/3$.
The main difference between P-uniform BPNC^i and $\mathsf{P{+}BPNC}^i$ is that in the latter, the intermediate circuit depends on x itself, rather than just on $|x|$. Thus, the Turing Machine transducer M computing C_x can potentially decide membership of x in L without even inspecting r, and simply produce a constant circuit. Also, there is no need for x itself to be given as input to C_x: if this were in any way advantageous, the Turing Machine M could simply substitute x inside the circuit. Finally, for inputs x_1, x_2 of the same size ($|x_1| = |x_2|$), the circuits produced by M can be very different. In contrast, the Turing Machine computing the circuit C_m only gets $|x|$ as input, so it must produce the same circuit.

Remark 12. Recall that BPP is not known to have complete problems. In fact, it would be considered as progress towards showing $\mathsf{P} = \mathsf{BPP}$ if one could prove such a thing. Contrast to promiseBPP which does have as complete the promise problem where the input is a circuit (promised to accept/reject bounded away from $1/2$) and the question is whether it accepts or rejects on the majority of the inputs. There is some superficial similarity of this complete problem with the definition of randomness compiler. However, the two objects are different, until one shows that $\mathsf{promiseBPP} = \mathsf{BPP}$.

By the discussion preceding Definition 10, it is easy to see that the logarithmic space randomized Stack Machine S with $p = 2^{O\left((\log n)^i\right)}$ passes over the random tape can simulate a Randomness Compiler with depth $O\left((\log n)^i\right)$. Our main contribution in this section is to show that, perhaps surprisingly, a partial converse is true.

Theorem 6. *For every positive integer i,*

$$\mathsf{P{+}BPNC}^i \subseteq \mathsf{BPPdL}\left[2^{O\left((\log n)^i\right)}\right] \subseteq \mathsf{P{+}BPNC}^{i+1}$$

The reason the second inclusion might be surprising is that, in a sense, we are able to formally

[4] We hope it is clear that in this statement, the word "random" refers to two standard, yet different concepts.

"break apart" (modulo the loss in the exponent) the computation that S performs on its input tape from the computation it performs on its random tape. Technically, the heart of this argument is (i) an adaptaption of the results in [Ruz80, Ruz81] and the observation that the constructions in these proofs are also efficiently computable, and (ii) the Time Compression Lemma 13, stating that given polynomial time precomputation depending *only on the input length*, we can "compress" the computation of a Stack Machine by giving it access to an advice tape. At first this seems unexpected since the computation of the Stack Machine depends on the input itself. Allender [All89] obtains a similar result, in a somewhat related setting with different parameters and proof. The formal statement of the Time Compression Lemma involves a somewhat technical extension of the Stack Machine model. We defer the precise statement and proof to Appendix C.

Lemma 13 (Time Compression Lemma (informal statement))**.** *Let $\mathcal{S}$ be a logarithmic space randomized Stack Machine that makes p passes over its random tape. Then, there exists a randomized Stack Machine $\mathcal{S}'$ which in addition has a polynomial time uniform advice tape (on which we do not count passes) such that $\mathcal{S}'$ decides the same as $\mathcal{S}$ and*

- *$\mathcal{S}'$ makes p passes over its random tape and*
- *$\mathcal{S}'$ runs in time $n^{O(1)}p$.*

For example, a randomized Stack Machine that works in exponential time and which makes polynomially many passes over its random tape can be simulated by a randomized Stack Machine with advice that works in polynomial time.

5 Discussion

To put things in perspective let us fix $s(n) = O(\log n)$; i.e. we consider the connection between *logarithmic space* Stack Machines and *polynomial time* Turing Machines. In this case, by parametrizing on the number of passes over the random tape we define a hierarchy of classes between P on the one side, and RP, BPP, NP, or PP on the other, depending on the error condition. The level p of this hierarchy is the class of languages decided by Stack Machines that are allowed p passes over the random tape (we think of $p = 0$ as representing no access to the random tape at all). We have completely settled the case of NP and PP: a single pass over the polynomially long random tape does not get us outside P, whereas two passes suffice to get the whole NP and PP.

Let us turn our attention to the more interesting case of bounded error. In this case, we do not know whether 2 passes over the random tape are useless (i.e., whether we can derandomize that class) or whether they give as significant a power jump as in the unbounded error case. However, we note that any potential derandomization argument, even for $p = 2$, would have to use the fact that the error is bounded (unless e.g. $\mathsf{P} = \mathsf{NP}$).

We find Theorem 6 paticularly interesting. It states that the two hierarchies defined between P and BPP, one through Randomness Compilers and one through Stack Machines, are essentially equivalent. A conceptual implication of this equivalence is that the hierarchies are not contrived.

It is natural to consider what type of questions one could hope to answer about randomness compilers short of derandomizing BPP. The standard construction of the circuit in the Cook-Levin theorem (see e.g. [DK00]) depends on the *input length*. Perhaps if the circuit is allowed to depend on the input itself (as in the case of a randomness compiler) one could reduce the depth of the standard construction. Of course, if $\mathsf{P} = \mathsf{BPP}$ then there is a randomness compiler where the circuits on the output are just trivial and constant. As another research direction, one could ask whether derandomizing $\mathsf{P{+}BPNC}^i$ (for some i) has implications towards the higher levels of P+BPNC.

We conclude by mentioning two technical points. The first regards an implication of Theorem 6. By Theorem 6 pseudo-random generators (PRGs) that fool NC^{i+1} circuits can be used to derandomize $\mathsf{BPPdL}\left[2^{O\left((\log n)^i\right)}\right]$. These PRGs can fool even non-uniform machines, which is common (e.g. [NW88]). That is, derandomizing using PRGs along BPNC (a class believed to be deeply inside P) we derandomize along BPPdL (which contains P).

The second point motivates the study of a new type of pseudo-random generators against *space bounded adversaries*. Before we proceed any further, we remark that the derandomization of the lower levels of the hierarchy, and in particular of $\mathsf{BPPdL}[\mathsf{polylog}] := \cup_{k>0}\mathsf{BPPdL}\left[\log^k n\right]$, may happen without the full derandomization of BPNC^2 when using pseudo-random generators (which is sufficient by Theorem 6). In particular, one can apply the Time Compression Lemma and extend the analysis in [Ruz81] to show (this is non-immediate) that $\mathsf{BPPdL}[\mathsf{polylog}]$ can be derandomized in quasi-polynomial time if there exists a pseudo-random generator that stretches a seed of length $\log^{O(1)} n$ to $n^{\Omega(1)}$ and it fools machines M, where: (i) M works in space $\log^{O(1)} n$, (ii) M is non-deterministic, (iii) the input bits are accessed in an arbitrary order (random access to the input), and (iv) each bit can be read at most $\log^{O(1)} n$ many times. In other words, if we construct stronger pseudo-random generators, or strengthen the analysis of existing ones (such as [Nis92] or [INW94]), which fool *space* bounded distinguishers, then we can derandomize $\mathsf{BPPdL}[\mathsf{polylog}]$, a probabilistic *time* class.

Acknowledgments

We'd like to thank Eric Allender, Allan Borodin, Mark Braverman, Stephen Cook, and Charles Rackoff for the useful discussions.

Appendix

A Additional notation and preliminaries

Stack Machines

We assume that a Stack Machine always halts on all inputs. Here are some additional conventions and useful definitions related to Stack Machines. We picture the stack vertically, so that symbols are always pushed and pop from the top. The *height* of the stack is defined to be the total number of symbols it contains. (So the empty stack has height 0.) We call a string placed on the random (external) tape the *external string*.

Definition 14 (Full configuration and surface configuration)**.** A *full configuration* (or just configuration) of a Stack Machine M consists of all information about M at a given time: the state M is in, the content of the work tape and its head, the entire content of the stack, the positions of the input tape and the external tape, and the current symbols on the input and external tapes. A *surface configuration* of M consists of all these pieces of information but instead of the stack content, it contains only the stack height and the top symbol on the stack.

Fact 15. *A Stack Machine M with space bound s and an external tape of size $2^{O(s)}$ can achieve stack height at most $2^{O(s)}$.*

Proof of Fact 15. Clearly, a surface configuration as defined above minus the stack height can be encoded on $c \cdot s$ bits, for some constant c. Imagine that every symbol ever placed on the stack is annotated with the surface configuration minus the stack height that caused it to be pushed on the stack. Assume that under some input and some random string, there is a point at which the stack height becomes $2^{c \cdot s} + 1$. This means one annotation is repeated. But then, M will never halt, because the computation between two successive equal annotations will be repeated indefinitely. This contradicts the assumption that M halts on all inputs and random strings. □

By Fact 15, there are $2^{O(s)}$ many surface configurations. We will use $C, C_1, C_2, \ldots$ for surface configurations, and $\overline{C}, \overline{C_1}, \overline{C_2}, \ldots$ for full configurations. For each full configuration $\overline{C}$ there is an unique surface configuration C, and we say that $\overline{C}$ *extends* C. (Potentially there are many full configurations that extend the same surface configuration.) For a (surface) configuration C, $\mathsf{head}(C)$ denotes the position of the external tape head in C, and $\mathsf{height}(C)$ denotes the stack height in C.

The following relation was considered in [Coo71].

Definition 16. Let M be a Stack Machine with 1 pass over the random tape. Let x be an input. For two surface configurations C_1, C_2 with the same stack height (i.e., $\mathsf{height}(C_1) = \mathsf{height}(C_2)$), and for a random string r of length $\mathsf{head}(C_2) - \mathsf{head}(C_1)$, we say that the pair (C_1, C_2) is *realized by* r if there exist full configurations $\overline{C_1}$ extending C_1 and $\overline{C_2}$ extending C_2 such that: when M is given input x, random string r is placed on the random tape starting at position $\mathsf{head}(C_1) + 1$, and M is started in configuration $\overline{C_1}$, M eventually arrives at a configuration $\overline{C_2}$, and the stack height never drops below the height in $\overline{C_1}$. We say that the pair (C_1, C_2) is *realizable* if it is realized by some r.

We say that a surface configuration C is *reachable* if there exists a random string r and a full configuration $\overline{C}$ extending C such that, if M is started in the initial configuration with input x and random string r, M eventually reaches $\overline{C}$.

Note that in the above definition, for a pair of surface configurations (C_1, C_2) it does not matter how they are extended to $\overline{C_1}$ and $\overline{C_2}$ as long as these extensions have the same stack content.

We make the convention that every transition of a stack machine is of exactly one of the following types: a *push*, a *pop*, an *external move* in which the external tape head moves, and an *(internal) move* for other all other transitions. When the external tape is nondeterministic (resp. random, or advice) we also call an external move a nondeterministic (resp. random, or advice) move.

Pseudo-random generators against $O(\log^i n)$-depth polynomial size circuits

Our PRGs definitions are more qualitative than usual. Several parameters have been fixed to reduce clutter. For example, we focus on PRGs that stretch polylogarithmic bits to polynomial. These parameters can be adjusted in the standard way to generalize our results. In what follows, all distinguishers are non-uniform circuits. We denote by U_n the random variable of the uniform distribution over $\{0, 1\}^n$.

Definition 17 (PRGs against adversaries with fixed complexity bounds)**.** Let $0 < \epsilon < 1$, $k \geq 1$. Let $G : \{0, 1\}^* \to \{0, 1\}^*$ be a function, such that $G(z)$ is computable in time $2^{O(|z|^{1/k})}$. We say that G is an $(\mathsf{NC}^i, k, \epsilon)$*-pseudorandom generator* if for every non-uniform NC^i circuit-family $\mathcal{C}$ and for sufficiently large $|z| := n$, $z \in \{0, 1\}^*$, where $|G(z)| = 2^{|z|^{1/k}} := m$

$$|\Pr[C_m(G(U_n)) = 1] - \Pr[C_m(U_m) = 1]| \leq \epsilon$$

where $C_m \in \mathcal{C}$ has m input bits.

B The unbounded case: Proof of Theorem 3

This theorem is proved by modifying Cook's argument [Coo71] . Let L be the language accepted by a randomized Stack Machine M: on input x, M makes at most one pass on any random string r (of length $2^{O(s)}$), and

$$x \in L \Leftrightarrow Pr_r[M \text{ accepts } (x, r)] > 1/2$$

To show $L \in \mathsf{P}$, we show that there is a $2^{O(s)}$-time algorithm that computes the number of random strings r that make M accept. We give a dynamic programming algorithm, which builds on the one used in [Coo71] to compute the realizability relation. Recall the notation of Section 2.

In what follows, we show how to compute, for every pair of realizable surface configurations (C_1, C_2), the exact number $\alpha(C_1, C_2)$ of random strings that realize (C_1, C_2). Then, to determine acceptance of x, we compute the sum over C_f of $\alpha(C_0, C_f)$ weighted by $2^{-\mathsf{head}(C_f)}$, where C_0 is the surface of the initial configuration, and C_f are surfaces of accepting configurations with different positions of the external tape head.

To compute α, we first define a partial order relation $\prec$ on pairs of surface configurations, as follows:

Definition 18. For two surface configurations C_1, C_2, we have $C_1 \prec C_2$ if either of the following holds:

(i) $\mathsf{head}(C_1) < \mathsf{head}(C_2)$; or

(ii) $\mathsf{height}(C_1) < \mathsf{height}(C_2)$; or

(iii) (C_1, C_2) is realizable and (C_2, C_1) is not realizable.

Since the original realizability relation is computable in time $2^{O(s)}$ by the original Cook's algorithm [Coo71], so is the relation $\prec$. If we restrict our attention to the set of all surface configurations with the same external head position and stack height, we know that realizability is transitive, so condition (iii) above effectively removes all cycles in the realizability relation (restricted to the same set.) This makes $\prec$ a partial order relation.

For a surface configuration C let $\mathsf{next}(C)$ denote the set of all possible surface configurations that can be obtained from C. Note that if the transition determined by C is not a pop or a move on the random tape, then $\mathsf{next}(C)$ consists of only one element which is completely determined by C. On the other hand, if the transition determined by C is either a move on the random tape, or a pop, then $\mathsf{next}(C)$ consists of two elements corresponding to the two possibilities for the symbol on the random tape, or the symbol popped from the stack. The next lemma provides us with the recursion for computing the table $\alpha(C_1, C_2)$.

Lemma 19. *For all surface configurations C_1, we have $\alpha(C_1, C_1) = 1$. For all surface configurations $C_1 \neq C_2$:*

- *Suppose that C_1 is followed by an internal move. Let $\{C_1'\} = \mathsf{next}(C_1)$. Then $\alpha(C_1, C_2) = \alpha(C_1', C_2)$.*
- *Suppose that C_1 is followed by a move on the random tape. Let $\{C_1', C_1''\} = \mathsf{next}(C_1)$. Then $\alpha(C_1, C_2) = \alpha(C_1', C_2) + \alpha(C_1'', C_2)$.*
- *Suppose that C_1 is followed by a push. Let x be the symbol that is pushed on the stack as dictated by C_1, and $\{C_1'\} = \mathsf{next}(C_1)$. Then*

  $$\alpha(C_1, C_2) = \sum_{C_3} \alpha(C_1', C_3) \cdot \alpha(C_4, C_2).$$

  where the sum is over all C_3 such that (C_1', C_3) is realizable and C_3 is followed by a pop, and C_4 is the next surface configuration of C_3 where x is the symbol popped from the stack.

Proof. The first two items are straightforward. For the last, we can treat the surface configurations C_1, C_2, C_4 as full configurations whose stacks are empty, and C_1', C_3 as full configurations whose stacks each contains only one symbol x. The sum on the RHS can be seen as summing over all possible first configurations C_4 reachable from C_1 that have the same stack height as C_1. Such configuration C_4 must be obtained from C_3 by a pop where the popped symbol is x. Because all configurations in the partial computation from C_1 to C_4 (except for C_1 and C_4 themselves) have stack height larger than that of C_1 and C_4, the total number of random strings that realize (C_1, C_4) in this way is precisely $\alpha(C_1', C_3)$. □

The following captures the interplay between the relation $\prec$ and the recursive formulas computing α.

Lemma 20. *Let C_1, C_2 be two surface configurations such that C_1 is reachable, and some entry of the form $\alpha(C_1, \cdot)$ depends on an entry of the form $(C_2, \cdot)$ in the formulas from Lemma 19. Then $C_1 \prec C_2$.*

Proof of Lemma 20. Observe that $(C_1, \cdot)$ never depends on $(C_2, \cdot)$ if either $\mathsf{head}(C_1) > \mathsf{head}(C_2)$ or $\mathsf{height}(C_1) > \mathsf{height}(C_2)$. Moreover, if either $\mathsf{head}(C_1) < \mathsf{head}(C_2)$ or $\mathsf{height}(C_1) < \mathsf{height}(C_2)$, then $C_1 \prec C_2$, so there is nothing to prove.

The remaining interesting case is where an entry $(C_1, \cdot)$ depends on an entry $(C_2, \cdot)$, and we have $\mathsf{head}(C_1) = \mathsf{head}(C_2)$ and $\mathsf{height}(C_1) = \mathsf{height}(C_2)$. Clearly, this happens when either C_1 is followed by a move transition directly yielding C_2, or C_1 is followed by a push yielding C_1', (C_1', C_2') is realizable, and C_2' is followed by a pop transition yielding C_2. In either case, (C_1, C_2) is realizable.

Assume that $C_1 \not\prec C_2$. By the discussion above, this can only happen if (C_2, C_1) is also realizable. But recall that C_1 is reachable. Then, M can get into an infinite loop by first reaching C_1, then repeating the computation between C_1 and C_2 indefinitely. This contradicts the assumption it halts on every input and every random string. □

Proof of Theorem 3. We compute $\alpha(\cdot, \cdot)$ row by row, as follows. Below, a linearization of a partial order is a total order that respects the partial order. Let Γ be the set of all surface configurations. We write $\Gamma_{i,j}$ for the set of surface configurations C with $\mathsf{head}(C) = i$ and $\mathsf{height}(C) = j$.

```
Initialize $\alpha(\cdot,\cdot) \leftarrow 0$
Compute $(\Gamma, \prec)$
For $i \leftarrow 2^{O(s)}$ down to 0
  For $j \leftarrow 2^{O(s)}$ down to 0
    Let $(\Gamma_{i,j}, \lhd)$ be a linearization of $(\Gamma_{i,j}, \prec)$
    For $C_1$ in $\Gamma_{i,j}$ in reverse order of $\lhd$
      For all $C_2$
        compute $\alpha(C_1, C_2)$ as in Lemma 19
```

Correctness follows from Lemma 20. By inspection, the running time is seen to be $2^{O(s)}$. □

C Randomness Compilers: omitted proofs

The proof of Theorem 6 is the most technically involved one. The non-trivial inclusion $\mathsf{BPPdL}\left[2^{\log^i n}\right] \subseteq \mathsf{P{+}BPNC}^{i+1}$ relies on (i) the Time Compression Lemma, (ii) adaptation of the results in [Ruz81, Ruz80], and by observing that the constructions in the modified proofs in [Ruz81, Ruz80] can be efficiently computed.

We first present the Time Compression Lemma.

Definition 21. A *log-space randomized Stack Machine with advice* is a randomized Stack Machine with three read-only tapes. (i) its input of length n, (ii) a $n^{O(1)}$-long random tape, and (iii) an advice tape whose content is computed in polynomial time on input 1^n.

We consider randomized Stack Machines with advice that are log-space bounded. As usual we only bound by p the number of passes over the random tape (in particular, there is no bound on the number of passes over the advice tape). Below, we restate the Time Compression Lemma.

Lemma 13. *Let $\mathcal{S}$ be a logarithmic space randomized Stack Machine that makes p passes over its random tape. Then, there exists a randomized Stack Machine with advice $\mathcal{S}_{\mathsf{adv}}$ such that $\mathcal{L}(\mathcal{S}_{\mathsf{adv}}) = \mathcal{L}(\mathcal{S})$ and*

- *$\mathcal{S}_{\mathsf{adv}}$ makes p passes over its random tape, and*
- *$\mathcal{S}_{\mathsf{adv}}$ runs in time $n^{O(1)}p$.*

Proof. It suffices to show that the computation between two successive input head-moves can be "compressed" to be polynomially long by the use of a P-uniform advice. Fix two arbitrary successive head-moves. Partition the computation γ between these head-moves in two phases. Suppose that immediately after reading the input symbol the stack level is at l. In phase 1, γ reaches its lowest stack height l_{min}. Let γ_1 be the computation subsequence of γ from the beginning until we reach the lowest stack level and just before we start going upwards (pushing symbols to the stack). Define γ_2 to be the complement of γ_1 wrt γ. Hence, in γ_2 the computation reaches its final stack height l_{last}. By a simple counting argument we have that the stack height of M is polynomial, and therefore the stack height in γ_1 gets decreased at most polynomially lower (from l to l_{min}) and in γ_2 gets increased po! lynomially higher (from l_{min} to l_{last}). We construct M' simulating M using the following P-uniform advice. The advice is a function from every surface configuration to the set of surface configuration together with two special symbols $\{\uparrow, \downarrow\}$. For every surface configuration σ define exactly one of the three pairs:

1. If starting from σ we can return to the same stack level without ever going below the initial stack-level (and without a head-move on the input) then consider the configuration after a maximally-long computation such that when M returns to the same stack-level the surface configuration is σ'. Then, the corresponding pair is (σ, σ').
2. If starting from σ we move at least one level upwards without ever returning to the initial stack-level (and without moving the head) then the pair is $(\sigma, \uparrow)$.
3. Else, the pair is $(\sigma, \downarrow)$.

Obviously, this is a well-defined function and we say that a surface configuration σ is of type (1), (2) or (3) respectively. Furthermore, using a variation of Cook's dynamic programming algorithm we have that we can compute this advice in polynomial time.

Between two successive head-moves M' simulates M by reading the advice tape and updating its surface configuration appropriately. In case of (1) it updates the worktape, the state and the top stack symbol. In case of (2) and (3) it simulates M for one step.

We refer to a *simulation step* as the computation sequence of M' in which M' reads the non-uniform tape, compares it to the current surface configuration and updates the surface configuration appropriately. In what follows the reader is reminded that γ_1, γ_2 is the computation of M which is simulated by the machine M', and that M' is given the non-uniform advice. We say that a function from the integers is 2-monotonically increasing (decreasing) if it is strictly increasing (decreasing) for two successive integers; i.e. for the function $h : \mathbb{Z}^+ \to \mathbb{Z}^+$, $h(n) \leq h(n+1)$ and $h(n) < h(n+2)$.

Claim 22. *In the simulation of γ_1 the stack height in M' is 2-monotonically decreasing. Hence, this simulation takes at most $2(l - l_{min})$ simulation steps of M'.*

Proof. Consider two successive stack levels $l_1 > l_2 := l_1 - 1$ in γ_1 and consider the first time M' gets to l_1. The current surface configuration σ_1 cannot be of type (2). Suppose that σ_1 is of type (2). Since we are in γ_1 we know that the stack level gets as low as l_{min}. If σ_1 is of type (2) then we know that during γ_1 the stack level will get back to l_1. Hence, σ_1 should instead be of type (1).

Hence, σ_1 is either of type (1) or of type (3). If it is of type (3) there is nothing left to show. Suppose σ_1 is of type (1). The fact that the next surface configuration in the simulation cannot be of the same type (1) follows by the maximality in the definition of type (1). □

Similarly, we show that in the simulation of γ_2 the stack height in M' is 2-monotonically increasing. □

Now, we are ready to give the proof of Theorem 6.

Proof of Theorem 6. P+BPNC^i $\subseteq$ $\mathsf{BPPdL}\left[2^{O(\log^i n)}\right]$ is the easy inclusion. Let $L \in \mathsf{P}$+BPNC^i and M be the polytime transducer that on input x computes C_x. Construct a Stack Machine M' that on input $\langle 1^{|x|}, k\rangle$ works as follows: use Cook's algorithm [Coo71] to simulate M so as to obtain the k-th output bit from the description of C_x. Note that each bit is computed without accessing the random tape. M' evaluates C_x on the provided randomness in the usual way (e.g. [Ruz81]) by a depth first search. Note that if the description of the circuit were given through oracle access, the evaluation procedure would have taken time $2^{O(\log^i n)}$. Hence, M' makes at most $2^{O(\log^i n)}$ reversals on the random tape and the accepting probability is the same as that of C_x.

To show $\mathsf{BPPdL}\left[2^{O(\log^i n)}\right] \subseteq \mathsf{P}$+$\mathsf{BPNC}^{i+1}$ we rely on Lemma 13 and on [Ruz80, Ruz81].

Let M' be a SM witnessing $L \in \mathsf{BPPdL}\left[2^{O(\log^i n)}\right]$, for an arbitrary such L. We will show how to construct a polytime M that on input x outputs a circuit C_x with the same accepting probability as M'.

There exists M'' extending M' as follows: M'' is M' as described in the proof of Lemma 13; i.e. it has an additional read-only advice tape and it works as specified in the proof of Lemma 13. Therefore, on input x given that the extra-tape contains this advice, M'' computes identically to M'. Syntactically, M'' is a SM with three read-only input tapes. When the 3rd tape contains the appropriate advice, M'' is a SM that works in space $O(\log n)$ and in time $2^{O(\log^i n)} n^{O(1)} = 2^{O(\log^i n)}$, $i \geq 1$. We assert the existence of an equivalent ATM M_{ATM} that works in space $O(\log n)$ and in time $O(\log^{i+1} n)$. It is straightforward to verify that all equivalences between SMs and ATMs in the constructions of Theorem 5 part 3 [Ruz81] p.379 (i.e. Theorem 2 [Ruz80] pp. 227-231), and Corollary 3 (c,d,e) [Ruz81] pp. 379-380 are the same when instead of one we have three inputs tapes. Hence, syntactically given the 3-input tape SM M'' we have an ATM M_{ATM} with 3-input tapes that computes identically. The constant description of M_{ATM} can be hardwired in a (polytime) TM M. Although M_{ATM} and M'' accept the same inputs, we are only interested in the computations where their 3rd tape contains the advice of Lemma 13; in which case M'' computes the same as M'. Intuitively, one can blur the distinction between space-time bounded ATMs and size-depth families of combinatorial circuits, and moreover we observe that given the description of the ATM we can construct efficiently the circuit for the corresponding input length. That is, the description of the polytime M should be evident through the observation that the construction in the proof of Theorem 3 [Ruz81] p.375 is computable in time $2^{O(S(n))} O(T(n)) n^{O(1)} = n^{O(1)}$, where $S(n) = O(\log n)$ and $T(n) = O(\log^{i+1} n)$ is the space and the time of M_{ATM}. For completeness we briefly review this construction below.

1. On input x use (the modified) Cook's algorithm to compute the advice of Lemma 13, which is a function of $n = |x|$.

2. M has hardwired the description of the ATM M_{ATM} and it computes the description of a circuit C_x. In this circuit, both the input x and the advice are hardwired using the constant

0/1 gates of the circuit.

3. The circuit gates are labelled with (α, t), where α is the configuration of the ATM, and t is the time, where the output gates has label $(\alpha_{\text{initial}}, 0)$, where α_{initial} is the starting configuration. Configurations of type $\forall, \exists$ correspond to gates $\wedge, \vee$, we connect gates (α, t), $(\beta, t+1)$ if α yields β. The only exceptions to this rule is when the time and the space becomes bigger than $T(n), S(n)$ in which case we hardwire the gates to 0, and when we have configurations accessing the input in which case instead of a gate we have an input gate.

Step (1) takes polynomial time. The construction of the circuit in Step 3, also takes polynomial time $(2^{O(S(n))}O(T(n))n^{O(1)})$. □

Derandomization of BPNC **using PRGs** $\implies$ **derandomization of** $\mathsf{BPPdL}\left[2^{\log^{O(1)} n}\right]$

Theorem 6 implies, through a standard textbook argument (see e.g. [AB09] the derandomization of $\mathsf{BPPdL}\left[n^{\mathsf{polylog} n}\right] := \mathsf{BPPdL}\left[n^{\log^{O(1)} n}\right]$ using PRGs that derandomize BPNC. For this we rely on PRGs introduced in Definition 17.

Corollary 23. *Let* $k, i \geq 1$. *Suppose that there exists a* $(\mathsf{NC}^{i+1}, k, \frac{1}{7})$*-PRG. Then,* $\mathsf{BPPdL}\left[2^{O(\log^i n)}\right] \subseteq \mathsf{Time}\left(2^{O(\log^k n)}\right)$.

Hence, if there exists a k for all i's then $\mathsf{BPPdL}\left[n^{\mathsf{polylog} n}\right] \subseteq \mathsf{Time}\left(2^{O(\log^k n)}\right)$, whereas if k is a function of i then $\mathsf{BPPdL}\left[n^{\mathsf{polylog} n}\right] \subseteq \mathsf{QuasiP}$, where $\mathsf{BPPdL}\left[n^{\mathsf{polylog} n}\right] := \cup_{k>0}\mathsf{BPPdL}\left[2^{\log^k n}\right]$ and $\mathsf{QuasiP} := \cup_{k>0}\mathsf{Time}\left(2^{\log^k n}\right)$.

Remark 24. We do not know how to show, without the use of PRGs, that the derandomization of BPNC implies the derandomization of $\mathsf{BPPdL}\left[n^{\mathsf{polylog} n}\right]$.

References

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009.

[All89] E. W. Allender. P-uniform circuit complexity. *J. Assoc. Comput. Mach.*, 36(4):912–928, 1989.

[BCD+89] A. Borodin, S. A. Cook, P. Dymond, L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SICOMP: SIAM Journal on Computing*, 18, 1989.

[Coo71] S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *J. Assoc. Comput. Mach.*, 18:4–18, 1971.

[DK00] Ding-Zhu Du and Ker-I Ko. *Theory of Computational Complexity*. John Wiley and Sons, 2000.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In Proceedings, *Symposium on Theory of Computing (STOC)'94*. 1994.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. *Comput. Complexity*, 13(1-2):1–46, 2004 (also STOC'03).

[KN97] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge University Press, New York, NY, USA, 1997.

[KV85] M. Karpinski and R. Verbeek. There is no polynomial deterministic space simulation of probabilistic space with a two-way random-tape generator. *Inform. and Control*, 67(1-3):158–162, 1985.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992 (also STOC'90).

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in logspace. *Theoret. Comput. Sci.*, 107(1):135–144, 1993 (also Structure in Complexity Theory'90).

[Nis94] N. Nisan. RL $\subseteq$ SC. *Comput. Complexity*, 4(1):1–11, 1994 (also STOC'92).

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. *J. Comput. System Sci.*, 49(2):149–167, 1994 (also FOCS'88).

[Ruz80] W. L. Ruzzo. Tree-size bounded alternation. *J. Comput. System Sci.*, 21(2):218–235, 1980.

[Ruz81] W. L. Ruzzo. On uniform circuit complexity. *J. Comput. System Sci.*, 22(3):365–383, 1981.

[Sak96] M. Saks. Randomization and derandomization in space-bounded computation. In *Proceedings, Conference on Computational Complexity (CCC)*, pages 128–149, 1996.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *J. Comput. System Sci.*, 43(2):380–404, 1991 (also STOC'87).