On Read-Once vs. Multiple Access to Randomness in Logspace

Noam Nisan *

June 15, 1995

Abstract

In the "correct" definition of randomized space-bounded computation, the machine has access to a random coin. The coin can be flipped at will, but outcomes of previous coin flips cannot be recalled unless they are saved in the machine's limited memory. In contrast to this read-once mechanism of accessing the random source, one may consider Turing machines which have access to a random tape. Here, the random bits may be multiply accessed by the machine.

In this note we show a very concrete sense in which multiple access to the random bits is better than read-once access to them: Every language accepted with bounded 2-sided error by a read-once-randomized Logspace machine, can be accepted *with zero error* by a randomized Logspace machine having multiple access to the random bits.

Finally we characterize the class of languages that can be accepted with two-sided error by randomized Logspace machines with multiple access to the random bits as exactly the class of languages that are in Logspace relative to almost every oracle.

1 Introduction

When defining randomized space-bounded computation, there are several possible variants to consider:

*Department of Computer Science, Hebrew University. Part of this work was done in MIT, supported by NSF 865727-CCR and ARO DALL03-86-K-017.

1. Type of error allowed:
   - Zero error: The machine gives an answer with probability of at least 1/2, and when it gives an answer the answer is always correct.
   - One-sided error: The machine always rejects any string not in the language, and accepts any string in the language with probability of at least 1/2.
   - Bounded two-sided error: The machine gives the correct answer with probability of at least 3/4.
   - Unbounded error: The machine gives the correct answer with probability strictly greater than 1/2.
2. Halting condition:
   - Always: The machine halts on any input and for any sequence of the random bits. It is easy to see that this condition is equivalent to that the worst case running time of the machine is at most exponential in the space bound.
   - Unconstrained: There is no requirement that the machine halt on every sequence of coin flips. It is possible for these kinds of machines to run for an expected time which is doubly exponential in the space bound. (Note: a condition requiring the machine to halt with probability one gives the same class. See e.g. [RST84].)
3. Access to randomness
   - Read-once: The machine may flip a random coin at any point. It cannot, however, recall the outcome of previous coin flips (unless, of course, they were explicitly saved in the limited space the machine has.)
   - Multiple access: The machine has a random tape with 2-way access to it. The machine may access any bit on this tape several times.

In this paper we focus our attention on this last issue: the access to randomness. For discussion of some of the other issues, and of the various complexity classes defined by the different choices, we refer the reader e.g. to [BCD*89, Gil77, BCP83, KV85, RST84, Hon80].

In this note we only consider machines which *halt always*, i.e. machines with running time which is at most singly exponential in the space bound. Randomized space S machines which are not required to do so may run for $exp(exp(S))$ time and are known to be quite strong, even if no error is allowed: in the case of read-once access to the random bits they have the full power of $NSPACE(S)$ ([Gil77, Imm88]); and in the case of multiple reads they have the full power of $DSPACE(exp(S))$ ([KV85]).

We denote the classes of languages which are accepted by *read-once* randomized *space*(S)-bounded machines by $ZPSPACE(S)$, $RSPACE(S)$, and $BPSPACE(S)$ (for zero error, one-sided error, and bounded two-sided error resp.); and the classes obtained by multiple access by $ZP^*SPACE(S)$, $R^*SPACE(S)$, and $BP^*SPACE(S)$. Most of our discussion will be focused on Logspace machines and the classes obtained by them: ZPL, RL, BPL, ZP^*L, R^*L, BP^*L (Note: in [BCD*89] the first three classes are denoted by $ZPLP$, RLP, and $BPLP$ resp.). Everything which we say, however, applies to any other "nice" (e.g. space constructible) space bound.

The read-once classes are the ones which are usually defined as the randomized analogs of Logspace, while multiple access to randomness is more similar to the kind of access to randomness given to randomized *circuits*. This access to randomness seems to be much stronger than read-once access. Consider for example the simulation of shallow circuits in Logspace: although it is known that $NC^1 \subset L \subset AC^1$, this result does not extend to show that $RNC^1 \subset RL$, indeed it is not known whether this is true (or even whether $RAC^0 \subset RL$?). The difficulty lies in the fact that multiple access to randomness is required for the simulation of RNC^1. Thus it can only be said that $RNC^1 \subset R^*L \subset RAC^1$. It is, in fact, reasonable to conjecture that RL does not contain RNC^1.

Consider also the deterministic simulation of randomized Logspace: it is well known that ZPL, RL, and BPL can all be simulated in polynomial time [BCP83]. It is not known, however, whether BP^*L, R^*L or even ZP^*L can be simulated in polynomial time (or indeed even in sub-exponential time). In fact, it is reasonable to conjecture that these classes are not contained in P.

It is interesting to note that, analogous to the case of randomized polynomial time computation, all of the above mentioned classes can be simulated

$non-uniformly$ in Logspace (using the techniques of [Adl78]). I.e. we have even for the largest class, BP^*L, that $BP^*L \subset L/poly$. In this paper we thus only discuss uniform simulations and classes.

In both definitions of access to randomness the obvious relations hold: $ZPL \subset RL \subset BPL$ and $ZP^*L \subset R^*L \subset BP^*L$, and also in both cases we have that $ZPL = RL \cap Co - RL$ and $ZP^*L = R^*L \cap Co - R^*L$. The relationships between the two types of classes have been, however, unknown.

We show here that even the weakest class with multiple access to the random bits, ZP^*L, contains all the classes with one-way access to randomness. We show that every language accepted by a read-once randomized Logspace algorithm with bounded two-sided error may also be accepted with *zero error* by a randomized Logspace machine with multiple access to the random bits.

Theorem 1 *$BPL \subset ZP^*L$.*

One immediate corollary regarding standard complexity classes is:

Theorem 2 *$BPL \subset ZPAC^1$.*

In fact, a slightly better simulation is implied, namely in the class $ZP-SAC^1$ (see, e.g., [BCD*89] for a definition). The best deterministic simulation of BPL known is $BPL \subset DET \subset NC^2$ due to [BCP83].

In [BNS89, Nis90] a different way in which multiple access to randomness is better than read-once access is demonstrated: Logspace algorithms using a polynomial number of random bits in a read-once fashion may be simulated by a randomized Logspace algorithm which uses only $O(log^2 n)$ random bits but with multiple access to them.

In the appendix to this paper we mention another characterization of the class BP^*L. We show that a language L is in BP^*L if and only if it can be recognized in Logspace relative to almost all oracles. Surprisingly, this characterization holds for the two different definitions of relativized Logspace (whether, or not, the query tape is also bounded by the space bound).

2 Main Result

We are given a BPL-machine M and an input w to it of length n. Our aim is to simulate the behavior of M on w in ZP^*L. In the discussion below the machine M and the input w are fixed. We assume w.l.o.g. that M uses at most space $c \log n$, runs for at most $m = m(n) = O(n^d)$ steps, and flips a random bit at each step. There are at most n^c configurations which M may be in (a configuration includes the total state of M (with input w): The contents of the work tapes, the locations of the heads on the read-only input tape, and the state of the finite control.)

The outline of the simulation is as follows: We find a small set of strings that are to be used in place of the random sequences of coin flips used by M. This set of strings has the property that the fraction of strings in the set which cause the machine to accept is approximately equal to the fraction of strings in $\{0,1\}^m$ which cause the machine to accept. Given this set, we simulate the behavior of M using each string in the set as random coin flips. We count the number of strings that cause M to accept, and thus derive a good approximation to the number of random sequences of coin flips which cause M to accept.

Definition 1 *Let $L \subset \{0,1\}^m$, and $S \subset \{0,1\}^m$. S is called ϵ-representative for L if*

$$|Pr_{s \in S}[s \in L] - Pr_{s \in \{0,1\}^m}[s \in L]| \leq \epsilon$$

In a sense the set S acts like the output of a pseudorandom generator relative only to this specific set L. It is not difficult to show that for every L there exists a small, polynomial size, set S which is ϵ-representative for L. In fact, a random S will do. The problem is to get our hands on a set which is *certified* to have this property. The main observation of this paper is that for sets L which represent sets of random strings which cause Logspace machines to accept, we can indeed get sets which are *certified* to be representative.

Definition 2 *Let v be a configuration of M and $0 \leq i \leq m$. $L_{v,i}$ is defined to be the set of strings in $\{0,1\}^m$ which when used by M as the sequence of coin flips result in v being the configuration reached after the i'th coin flip. $L^0_{v,i}$ (resp. $L^1_{v,i}$) is defined to be the sets of strings in $L_{v,i}$ which have 0 (resp. 1) in the $i+1$'st position.*

Definition 3 *A set $S \subset \{0,1\}^m$ is called ϵ-good if for every $0 \leq i < m$ and every configuration v:*

$$||L^1_{v,i} \cap S| - |L^0_{v,i} \cap S|| \ \leq \ \epsilon |S|$$

We show that (1) a randomly selected set of polynomial size S is very likely to be good; (2) A Logspace machine can test and verify whether a set S is good; and (3) a set which is good is also representative for the set of sequences of coin flips which cause the machine to accept.

Lemma 1 *Let $\epsilon = n^{-e}$ then there exists a constant $b = b(c, d, e)$ such that if S is a set of size n^b chosen at random in $\{0,1\}^m$ then $Pr[S\, is\, not\, \epsilon - good] \leq 1/2$.*

Proof: By proper choice of b $(= O(c + d + e))$, and using the Chernoff bounds, it is easy to show that for any fixed v, i the probability that

$$||L^1_{v,i} \cap S| - |L^0_{v,i} \cap S|| \ \geq \ \epsilon|S|$$

can be bounded from above by $n^{-(c+d)}/2$. Adding up these probabilities for all pairs of i and v completes the proof of the lemma. □

Lemma 2 *The following problem can be decided in Logspace:*

INPUT: A set $S \subset \{0,1\}^m$, a rational number ϵ.

OUTPUT: Is S ϵ-good?

Proof: The following is a Logspace algorithm for the problem. We run over all possible choices of v and i. For each choice we run over all strings $s \in S$ and simulate M on input w with s as the outcome of the random coin flips. Doing this, we count the sizes of $L^0_{v,i}$ and $L^1_{v,i}$ and compare the difference to the product $\epsilon|S|$. We accept if for every v and i the difference was at most $\epsilon|S|$.

Note that this simulation is in Logspace as: (1) M can be simulated in Logspace (2) there are only polynomially many choices of i and v. □

Lemma 3 *If S is ϵ-good then it is $(\epsilon m n^c)$-representative of the set of sequences of random bits that cause M to accept.*

To carry out the induction we will prove a stronger lemma.

Lemma 4 *Let S be ϵ-good then for any $0 \leq i \leq m$ we have that*

$$\sum_v |Pr_{s \in S}[s \in L_{v,i}] - Pr_{s \in \{0,1\}^m}[s \in L_{v,i}]| \ \leq \ n^c i \epsilon$$

Where the sum is taken over all configurations v of M.

Proof: The proof is by induction on i. For $i = 0$ the statement is trivial. Assume the lemma for $i-1$ and prove for i: For a configuration v let U_v^0 (resp. U_v^1) be the set of configurations u such that a coin is flipped in u, and if the outcome of the coin is 0 (resp. 1) then M's new configuration is v. By this definition it is clear that $L_{v,i}$ is given by:

$$L_{v,i} = \bigcup_{u \in U_v^0} L_{u,i-1}^0 \cup \bigcup_{u \in U_v^1} L_{u,i-1}^1$$

where the union is of disjoint sets.

Now fix a configuration v and consider

$$|Pr_{s\in S}[s \in L_{v,i}] - Pr_{s\in\{0,1\}^m}[s \in L_{v,i}]|$$

This can be bounded from above by

$$\sum_{u\in U_v^0} |Pr_{s\in S}[s \in L_{u,i-1}^0] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}^0]| +$$

$$\sum_{u\in U_v^1} |Pr_{s\in S}[s \in L_{u,i-1}^1] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}^1]|$$

The fact that S is ϵ-good implies that for each u:

$$|Pr_{s\in S}[s \in L_{u,i-1}^0] - \frac{1}{2}Pr_{s\in S}[s \in L_{u,i-1}]| \leq \frac{\epsilon}{2}$$

and similarly

$$|Pr_{s\in S}[s \in L_{u,i-1}^1] - \frac{1}{2}Pr_{s\in S}[s \in L_{u,i-1}]| \leq \frac{\epsilon}{2}$$

We also have that

$$Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}^0] = Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}^1] =$$

$$= \frac{1}{2}Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}]$$

Thus for any u we can bound:

$$|Pr_{s\in S}[s \in L_{u,i-1}^0] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}^0]| \leq$$

$$\frac{1}{2}|Pr_{s\in S}[s \in L_{u,i-1}] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}]| + \frac{\epsilon}{2}$$

(and similarly for $L^1_{u,i-1}$.)

We thus get that for each configuration v:

$$|Pr_{s\in S}[s \in L_{v,i}] - Pr_{s\in\{0,1\}^m}[s \in L_{v,i}]| \leq$$

$$\frac{1}{2}\sum_{u\in U_v^0}\Big(|Pr_{s\in S}[s \in L_{u,i-1}] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}]| + \epsilon\Big) +$$

$$\frac{1}{2}\sum_{u\in U_v^1}\Big(|Pr_{s\in S}[s \in L_{u,i-1}] - Pr_{s\in\{0,1\}^m}[s \in L_{u,i-1}]| + \epsilon\Big)$$

Summing up over all configurations v, and recalling that the sets U_v^0 are a partition of the configurations, and similarly the sets U_v^1, allows us to apply the induction hypothesis to this last bound and conclude the proof of the lemma. □

The conjunction of lemmas 1, 2 and 3 yields the simulation of BPL in ZP^*L.

Theorem 1 $BPL \subset ZP^*L$.

Proof: The ZP^*L machine will simulate the behavior of the given BPL machine M on the given input w as follows: The random tape is used to supply a random set S of size n^b of strings of length m (where b is the constant given in lemma 1). It is then verified that the set is indeed good using the algorithm in lemma 2 – in the unlikely case that it is not good we halt without giving an answer.

If the set is verified to be good then by lemma 3 we know that it is representative for the set of sequences of coin flips which cause M to accept; so we simulate M using on each string in S as coin flips, count the number of acceptances, and thus can decide whether the fraction of sequences of coin flips causing the machine to accept is below 1/3 or above 2/3. This gives the information of whether the the machine M accepted the input w or not. It is clear that this simulation is in ZP^*L. □

Since $ZP^*L \subset ZPAC^1$ we also get

Theorem 2 $BPL \subset ZPAC^1$

3 Appendix: Logspace relative to a random oracle

There are (at least) two different reasonable definitions of relativized Logspace that are used in the literature. In the first (weak) definition, the length of the query tape is also limited by the space bound, and thus all queries must be of length $O(\log n)$. In the second (strong) definition the space bound does not apply to the query tape, and thus queries can have polynomial length. It is not difficult to see that for almost all oracles (in the sense of measure) these two notions give different classes.

Here we look at the class of languages that can be decided in Logspace relative to almost all oracles.

Definition 4 *For a complexity class C we define the class $almost-C$ to be the class of languages L such that*

$$Pr[L \in C^A] = 1$$

where A is an oracle chosen at random. For C = Logspace we distinguish between which definition of randomized Logspae was used and get the two classes $almost_w - L$ (weak definition) and $almost_s - L$ (strong definition).

The power of $almost - C$ is known for many complexity classes C, e.g. $almost{-}P = BPP$ ([BG81]), $almost{-}NP = AM$ ([NW88]), $almost{-}PH = PH$ ([NW88]). Here we determine what is $almost - L$. Surprisingly, it turns out that both definitions of relatived Logspace give the same class.

Theorem 3

$$almost_w - L \; = \; almost_s - L \; = \; BP^*L$$

Proof: We show that (1) $BP^*L \subset almost_w - L$, and (2) $almost_s - L \subset BP^*L$.

Proof of (1): This part is rather standard. Given a BP^*L Turing machine M that accepts a language L, and a constnat $\epsilon > 0$ we build a Logspace oracle Turing machine M' that accepts L relative to a $1 - \epsilon$ fraction of the oracles. First amplify the probability of success of M to $\epsilon 4^{-n}$ for every input of length n, then M' will use the i'th bit of the oracle each time M reads the i'th random bit from the random tape. It follows that for each input of length n, M' will be correct on that input for all but $\epsilon 4^{-n}$ fraction of oracles. Summing over all possible inputs we get that M' errs on even a single input for at most ϵ fraction of oracles.

Proof of (2): We start with the usual manuevers. Given a language L in $almost_s - L$ we have that for measure 1 of oracles A there exists a Logspace Turing machine M such that M^A accepts L. Since there are only countably many Turing machines, some Turing machine M' accepts L in $Space(c \log n)$ relative to ϵ fraction of oracles for some $\epsilon > 0$, and integer c. Using the Lebesgue density theorem we see that for some fixed initial segment of oracle entries, M' accepts L in $Space(c \log n)$ relative to 2/3 measure of oracles with that prefix. Finally, we can hardwire this prefix and obtain an oracle Turing machine M that accepts L in $Space(c \log n)$ relative to 2/3 measure of oracles A. We will simulate M in BP^*L.

To this point everything was standard, The natural thing to do next is to simulate M replacing each oracle query by reading a random bit. The difficulty with this is that queries to the oracle may have long ($2^{c \log n} = n^c$) length, while in BP^*L we only have access to a polynomial number of random bits accessed by $O(\log n)$-bit long addresses. This difficulty will be solved by using a generator. The generator will be fed a polynomial number of random bits, and will answer any query (of length n^c) by a bit. We will describe a generator G that (1) can be computed by a Logspace machine and (2) can be used to run M, without changing the acceptance probability (from that given by a truly random oracle).

G will be a generator which is n^c-wise independent. It accepts two parameters: an n^c-bit long query q and an n^d-bit random seed x, and outputs a single bit (d is a constant that depends on c). It satisfies the following property: For any sequence of queries $q_1, \ldots, q_{n^c}$, and any sequence of values $\epsilon_1, \ldots, \epsilon_{n^c} \in \{0, 1\}$ we have that

$$Pr_x[G(q_1, x) = \epsilon_1 \; and \; \ldots \; and \; G(q_{n^c}, x) = \epsilon_{n^c})] = 2^{-n^c}$$

It is an easy observation that our machine M will accept any string with the same probability whether it is given a truly random oracle A or the output of G where the random seed is chosen at random. This is so since for any sequence of queries $q_1, \ldots, q_{n^c}$, and any sequence of answers $\epsilon_1, \ldots, \epsilon_{n^c} \in \{0, 1\}$, we are assured that the probability that A accepts after getting this sequence of answers to this sequence of queries is the same both cases (A cannot make more than n^c queries). It is left to construct such a generator G in Logspace.

We use the construction from [NW88]. It should be noted that the reason that this construction works is different from the reason that the construction works in [NW88]; here we only require that it is n^c-wise independent and that it is easy to compute. Our random seed x is a p by p array of bits where p is the smallest prime number greater than n^{2c}. Thus the random seed is represented as $\{x_{ij} | 0 \leq i, j < p\}$. The value of $G(q, x)$ is computed as follows: q is treated as representing a polynomial of degree $n^c - 1$ over $GF(p)$, where the i'th bit of q is taken to be the i'th coefficient of the polynomial. The value of $G(q, x)$ is taken to be the parity of all bits of the random seed of the form $x_{i,q(i)}$ where $0 \leq i < p$.

We first prove that G is n^c-wise independent. Let $q_1, \ldots, q_{n^c}$ be a sequence of queries and consider the functions $l_i(x) = G(q_i, x)$ defined by them. These function are clearly linear functions over $GF(2)$, we claim that they are independent (this clearly suffices to show that G is n^c-wise independent). Consider any two such functions l_i and l_j and look at the variables that are common to both. There are at most n^c such variables since any two polynomials of degree $n^c - 1$ can agree on at most n^c values. Thus for any i, l_i cannot be represented as a sum (over $GF(2)$) of less than $p/n^c > n^c$ other l_j's. The computation of G can be done in Logspace as p is $O(\log n)$ bits long and thus can be found in Logspace, and arithmetic $mod\, p$ can also be done in Logspace. The rest of the operations needed to compute G are trivially in Logspace. □

4 Acknowledgements

I'd like to thank Mike Sipser for pointing out to me, some time ago, the distinction between one-way and multiple access to randomness in space-bounded computation, as well as many of the facts mentioned in the introduction. I'd also like to thank Russell Imagliazzo and Avi Wigderson for many helpful discussions, and Walter Ruzzo for several helpful suggestions.

References

[Adl78] L. Adleman. Two theorems on random polynomial time. In 19^{th} *Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan*, pages 75–83, 1978.

[BCD*89] A. Borodin, S.A. Cook, P.W. Dymond, W.L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM J. Comput.*, 18(3):559–578, 1989.

[BCP83] A. Bordin, S. Cook, and N. Pippenger. Parallel computation for well-endowed rings and space-bounded probabilistic machines. *Information and Control*, 58:113–136, 1983.

[BG81] C.H. Bennett and J. Gill. Relative to a random oracle a, $p^a \neq np^a \neq co - np^a$ with probability 1. *SIAM J. Comput.*, 10:96–113, 1981.

[BNS89] L. Babai, N. Nisan, and M. Szegedy. Multiparty protocols and logspace-hard pseudorandom sequences. In *Proceedings of the* 21^{st} *Annual ACM Symposium on Theory of Computing, Seattle, Washington*, pages 1–11, 1989.

[Gil77] J. Gill. Computational complexity of probabilistic turing machines. *SIAM J. Comput.*, 6:675–695, 1977.

[Hon80] J.W. Hong. On similarity and duality of computation. In 21^{th} *Annual Symposium on Foundations of Computer Science, Syracuse, New York*, pages 348–359, 1980.

[Imm88] N. Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comp.*, 17:935–938, 1988.

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

[Nis90] N. Nisan. Psuedorandom generators for space-bounded computation. In *Proceedings of the* 22^{st} *Annual ACM Symposium on Theory of Computing*, 1990.

[NW88] N. Nisan and A. Wigderson. Hardness vs. randomness. In 29^{th} *Annual Symposium on Foundations of Computer Science, White Plains, New York*, pages 2–12, October 1988.

[RST84] W.L. Ruzzo, J. Simon, and M. Tompa. Space-bounded hierarchies and probabilistic computation. *J. Comp. and Syst. Sci.*, 28(2):216–230, 1984.