Improved deterministic time simulation of nondeterministic space for small space: a note

Ee-Chien Chang Chee Yap
Courant Institute of Mathematical Sciences
New York University
251 Mercer Street
New York, NY10012

August 1, 1994

Abstract

We show that $NSPACE(s(n)) \subseteq DTIME(n \cdot O(1)^{s(n)})$. This improves the known bound of $NSPACE(s(n)) \subseteq DTIME(n^2 \cdot O(1)^{s(n)})$ when the space is "small", namely, $s(n) = o(\log n)$. We use a simple encoding trick combined with an amortization argument.
Key words: computational complexity; theory of computation

Introduction

The known non-deterministic space bound in terms of deterministic time is $NSPACE(s(n)) \subseteq DTIME(n^2 \cdot O(1)^{s(n)})$ (see [1]). Our basic definitions are standard but may be found in [1]. In particular, we assume the multi-tape Turing machine in which the *input tape* is read-only and space used on the *input tape* is not counted. Furthermore, we assume that the time or space bounds for a Turing machine apply only when an input is accepted. So for rejected inputs, the time or space usage could be infinite. This is sometimes called the "strong mode" of acceptance. Note that we make no assumption on the complexity function $s(n)$. In particular, $s(n)$ is not required to be constructible although the result is only of interest when $s(n) = o(\log n)$ and $s(n) = \Omega(\log \log n)$.

The idea in previous proofs is to have the simulating deterministic machine list down all the possible configurations of the simulated non-deterministic machine, and then mark all the configurations derivable from the initial configuration. We improve the bound by having a compact encoding of configurations, combined with an amortization argument. Our bound is optimal for any simulation that must visit all configurations of space $s(n)$ since there are $n \cdot O(1)^{s(n)}$ such configurations.

Theorem 1 *For any complexity function $s(n)$, $NSPACE(s(n)) \subseteq DTIME(n \cdot O(1)^{s(n)})$.*

Proof. Let N be a nondeterministic machine with k work tapes accepting in space s. Without lost of generality, we may assume $k = 1$. We construct a deterministic M with 2 work tapes that accepts in time $n \cdot O(1)^{s(n)}$ the language accepted by N. As in [1], M begins by listing down on one work tape all the possible configurations of N and then marking all the configurations derivable from the initial configuration. If an accepting configuration is ever marked, then M accepts at once. Otherwise, M does not halt.

We proceed to give the details for marking derivable configurations. We refer to M's work tapes as "tape 1" and "tape 2". Most actions occur on tape 1, the use of tape 2 is normally only inferred.

For any $h \geq 1$, let C^h be a listing, in some fixed but otherwise arbitrary order, of all words of the form

$$D = a_1 a_2 \cdots a_{i-1} q a_i \cdots a_h, \quad (1 \leq i \leq h)$$

where q is a state and the a_j's belongs to the tape alphabet of N. Each D in C^h corresponds to a usual configuration of N except that the input word and the input head position is not encoded. We call D a *storage configuration.* Note that the a_j's may be blank symbols and the position of q here indicates that the work head of N is scanning tape symbol a_i in D.

On any input x of length n, M computes in stages: in stage h, $h \geq 1$, M simulates the actions of N restricted to configurations of N using space at most h. To do this, M first lists n copies of C^h consecutively onto tape 1. This is easy to do with the help of tape 2 and assuming that C^{h-1} is available (for $h > 1$). Let C_i^h be the i-th copy of C^h, and call C_i^h the i-th *block*. If a storage configuration D occurs in the i-th block, then D represents an ordinary configuration whose input head is at the i-th position. The i-th input symbol is easily determined during the simulation by requiring M's input head to be in the i-th position whenever the work head is reading within the block C_i^h. Note that the space to store each block is $h \cdot O(1)^h = O(1)^h$. Since there are n blocks, the total space is $n \cdot O(1)^h$.

On a separate track of tape 1, we mark below each configuration as having one of the three statuses: *unseen*, *visitable*, *visited*. The simulation of N is now described using the following pseudo-code.

COMMENT: The counter $curr$ contains the index i of the block C_i^h containing the most recently visited configuration. $curr$ is not explicitly maintained by M, but is introduced to facilitate discussions.

INPUT is a word x;

$h \leftarrow 1$;

LOOP A

    List $C_1^h, C_2^h, \cdots, C_n^h$ on tape 1;

    Mark all configurations as *unseen*;

    Mark the starting configuration (which is in block 1) as *visitable*;

    $curr \leftarrow 1$;

    LOOP B

        COMMENT: Search for the leftmost *visitable* configuration, D.

    (1) Move the work head to the beginning of $C^h_{\max(1,curr-1)}$;

    (2) Search rightward for the first *visitable* configuration D;

            Exit LOOP B if no such configuration is found;

            Exit LOOP A if D is an accepting configuration;

    (3) Mark D as *visited*;

    (4) Update $curr$ to be the index of the block containing D;

    (5) Search for all the *unseen* configurations in $C^h_{curr-1}, C^h_{curr}, C^h_{curr+1}$

            that are derivable from D in one step; mark any such

            configurations *visitable*;

    (6) Move the work head back to D;

    end LOOP B;

    $h \leftarrow h + 1$;

end LOOP A;

Halt and accept x.

Our goal is to bound the time for a halting computation. Let "iterations" refer to loops corresponding to LOOP B, and "stages" refer to loops of LOOP A. Consider stage h. Let b $(= O(1)^h)$ be the size of a block, and let m $(= n \cdot O(1)^h)$ be the number of configurations of N using space h. Then there are at most m iterations. During an iteration, with the exception of step (2), each of steps (1)-(6) takes $O(b)$ time. For instance, it is easy to see that step (5) takes time $O(b)$, if we first store a copy of D on tape 2. Now consider step (2), which searches the blocks in the order $C^h_{curr-1}, C^h_{curr}, C^h_{curr+1}, \cdots$. It is easy to see that, inductively, no *visitable* configuration occurs in blocks C^h_j where $j < curr - 1$. Therefore, this search will not miss any *visitable* configuration (this leads to the correctness proof for this simulation). In the worst case, step (2) may take time $O(nb)$. This gives a time bound of $O(nb)$ per iteration, and $O(nbm) = n^2 \cdot O(1)^h$ for stage h. However, this bound is not tight. We argue that the "amortized cost" per iteration is only $O(b)$. Here is the key observation: the number of times that any given block is scanned by step (2) is at most *one more* than the number of times it is scanned by step (1). But step (1) visits at most m blocks in stage h. The key observation then implies that step (2) visits at most $2m$ blocks in stage h. Amortized (or average) over the m iterations, step (2) visits at most 2 blocks per iteration. Thus each iteration takes amortized time $O(b)$, and stage h takes overall time $O(mb) = n \cdot O(1)^h$.

Now, an input x is accepted by N iff N accepts x at some stage h, $h \leq s(|x|)$. It follows that the total time to accept an input of length n is

$$\sum_{h=1}^{s(n)} n \cdot O(1)^h = n \cdot O(1)^{s(n)}.$$

□

Remark. Our simulation will not halt if the input x is not accepted by N. This exploits the "strong acceptance mode" convention. If we insist that the time bound $O(1)^{s(n)}$ must hold even when x is not accepted, then it suffices to replace LOOP A by a FOR-LOOP that counts to $s(|x|)$ stages (and rejecting if it does not accept inside the FOR-LOOP). In this case, we require that $s(n)$ could be computed in $n \cdot O(1)^{s(n)}$ time. This is slightly weaker than requiring $s(n)$ to be space-constructible.

References

[1] C.K. Yap. *Theory of Complexity Classes*. Oxford University Press. To appear and available by anonymous ftp to cs.nyu.edu (cd pub/local/yap).