Theoretical Computer Science 10 (1980) 335–338
© North-Holland Publishing Company

NOTE

HALTING SPACE-BOUNDED COMPUTATIONS*

Michael SIPSER
Computer Science Division, University of California, Berkeley, CA 94720, U.S.A.

Communicated by Richard Karp
Received January 1979
Revised July 1979

The main result of this paper is that for any deterministic Turing machine which runs in space $S(n)$ and which possibly rejects by looping, there is an equivalent Turing machine which runs in the same amount of space and never loops. Hopcroft and Ullman [2] have previously shown this for $S(n) \geq \log n$ but conjecture that it is false for small S. Their proof for $S(n) \geq \log n$ counts steps on a separate track of the tape and shuts the machine off if it has run for too long. Hartmanis and Berman [1] prove this for $S(n) < \log n$ on unary languages with a more complicated counting technique.

In addition to space bounded Turing machines, our technique applies to other deterministic computation models. Using it, we can enforce termination on two-way finite automata without exponentially increasing the number of states and on two-way multihead finite automata without increasing the number of heads. The analogous questions about non-deterministic computation, however, remain open.

These results yield simpler, more general, and tighter diagonalization arguments. For example, we can now remove the log n restriction on the Stearns, Hartmanis and Lewis [10] proof of the space hierarchy theorem. Also, we can directly obtain Ibarra's result for multihead automata that $k+2$ heads are better than k [4].

We use Turing machines as formalized in [3]. A language A is *accepted in $S(n)$ space* if a machine accepts A and never visits more than $S(n)$ worktape squares for all inputs of length n. By a *configuration* of a machine we mean the entire status of the machine at a point in time, including the worktape contents, positions of both heads, and state of the finite control. Without loss of generality we require that the machine, when accepting an input, compeletely erases its worktape, returns the heads to their starting points and then halts in the unique accepting state.

* This research was supported by NSF grant MCS75-21760-A01 and an IBM Graduate Fellowship.

Theorem 1. *For every Turing machine M there is a Turing machine N such that on all input strings x:*

(1) N accepts iff M accepts,

(2) N uses no more tape than does M,

(3) N does not loop on a finite amount of tape.

Proof. First we present a procedure for determining whether M's starting configuration reaches the accepting configuration by a computation which uses at most space k. Consider the configurations of M using at most space k to be the nodes of a finite directed graph. Because M is deterministic, the component of this graph which contains the accepting configuration is a tree rooted at the accepting configuration. Our procedure begins at the accepting configuration and performs a depth-first search of this tree, to determine if the starting configuration is a member.

This search procedure can be easily implemented on a Turing machine N using space k. When N is visiting a node, it's worktape contains a copy of M's worktape and it's read head is in the same place as M's read head. Arcs are traversed forward by running M forwards one step. Arcs are traversed backward by running M backwards one step. At some point there may be several valid ways to run M backwards. These are ordered in some predetermined way and N selects them one at a time in the usual depth-first search manner. If, however, there are no valid ways to run M backwards, either because the current configuration has no predecessor or because the predecessors use space $k+1$, then a leaf of the tree has been reached and the search instead proceeds forwards.

A difficulty arises in utilizing the above reverse search procedure because N has no advance knowledge of k, the amount of space used by M. This is overcome by running the procedure successively for $k = 1, 2, 3, \ldots$ until an accepting computation is found. This uses exactly as much space as does M, provided M accepts. However, if M does not accept, then N will run forever. So instead, N ensures that M uses at least space $k+1$ before embarking on a $k+1$-search. It does so by cycling through all configurations using space k and selecting those which are about to use a $k+1$st square on the next step. Form each of these configurations it performs a reverse search for the start configuration. If none is successful then N rejects since it knows that M does not use space $k+1$. If, on the other hand, one of these searches succeeds, then N performs a $k+1$-search from the accepting configuration, as before.

Corollary 1. *All deterministic Turing machine space classes are closed under union, intersection and complementation.*

deterministic

Theorem 2. *For every n-state two-way finite automaton, there is an equivalent* $O(n^2)$*-state two-way finite automaton which always halts.*

Proof. Modify the automaton to accept in a unique state at the right end of the tape.

Then, a single reverse search from the accepting configuration can be used to determine acceptance.

Corollary 2. *The class 2D* [9] *is closed under union, intersection and complementation.*

Meyer and Winklemann [5] have observed that if the size of the alphabet is held fixed in the above theorem, then an $O(n)$ state, always halting 2dfa can be obtained. This follows because, to carry out the reverse search procedure, it is sufficient for the automaton to remember only the state of the current node, whether the simulation is currently going forwards or backwards, and the contents of the tape within two squares of the head. The simulation itself is identical, except that the two-step sequence of going from a node to its successor and then the next predecessor, must be shortcut into a single step. We omit the details.

Theorem 3. *For every k-head two-way finite automaton, there is an equivalent k-head two-way finite automaton which always halts.*

Proof. Same as above.

Corollary 3. *There are languages which can be accepted by a $k+2$-head two-way finite automaton and which cannot be accepted by any k-head two-way finite automaton.*

Proof. By diagonalization.

This result originally appears in [4] and, as pointed out by Seiferas, has been recently strengthened by Monien [6, 7] who shows that $k+1$-heads are better than k, even for unary languages [8]. These proofs use 'transformational' methods that are more difficult than the straightforward diagonalization which we use.

Acknowledgment

I gratefully thank my advisor, Manuel Blum, for many valuable and enjoyable discussions. This paper probably would not have existed without them.

References

[1] J. Hartmanis and L. Berman, On tape bounds for single letter alphabet language processing, *Theoret. Comput. Sci.* **3** (1977) 213–224.

[2] J.E. Hopcroft and J.D. Ullman, Some results on tape bounded Turing machines, *J. ACM* **16** (1967) 168–177.

[3] J.E. Hopcroft and J.D. Ullman, *Formal Languages and Their Relation to Automata* (Addison-Wesley, Reading, MA, 1969).
[4] O.H. Ibarra, On two-way multihead automata, *J. Comput. System Sci.* **7** (1973) 28–36.
[5] A.R. Meyer and K. Winklemann, personal communication.
[6] B. Monien, Transformational methods and their application to complexity problems, *Acta Informat.* **6** (1976) 95–108.
[7] B. Monien, Corrigendum: Transformational methods and their application to complexity problems, *Acta Informat.* **8** (1977) 383–384.
[8] B. Monien, Hierarchies for unary languages, to appear.
[9] W.J. Sakoda and M. Sipser, Nondeterminism and the size of two-way finite automata, *Proc. Tenth Annual ACM Symposium on Theory of Computing* (1978).
[10] R.E. Stearns, J. Hartmanis and P.M. Lewis II, Hierarchies of memory limited computations, IEEE Conference Record on Switching Circuit Theory and Logical Design (1965) 179–190.