INFORMATION AND CONTROL 55, 20–39 (1982)

On Efficient Simulations of Multicounter Machines*

PAUL M. B. VITÁNYI

Mathematisch Centrum, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands†

An oblivious 1-tape Turing machine can simulate a multicounter machine on-line in linear time and logarithmic space. This leads to a linear cost combinational logic network implementing the first n steps of a multicounter machine and also to a linear time/logarithmic space on-line simulation by an oblivious logarithmic cost RAM. An oblivious $\log^* n$-head tape unit can simulate the first n steps of a multicounter machine in real-time, which leads to a linear cost combinational logic network with a constant data rate.

1. INTRODUCTION

In many computations it is necessary to maintain several counts such that, at all times, an instant signal indicates which counts are zero. Keeping k counts in tally notation, where a count is incremented/decremented by at most an unit in each step, governed by the input and the set of currently zero counts, is formalized in the notion of a *k-counter machine* or k-CM, Fischer *et al.* (1968). Multicounter machines have been studied extensively, because of their numerous connections with both theoretical issues and more or less practical applications. The purpose of this paper is to investigate the dependence of the required time and storage, to maintain counts, on storage structure and organization, and the cost required by a combinational logic network. To do this, we use a notion of auxiliary interest: that of an oblivious Turing machine. An oblivious Turing machine is one whose head movements are fixed functions of time, independent of the inputs to the machine, cf. Pippenger and Fischer (1979). The main result obtained here shows that an oblivious Turing machine with only *one* storage tape can simulate a k-counter machine on-line in linear time and in storage logarithmic in the maximal possible count. These bounds are optimal, up to order of magnitude, also for on-line simulation by nonoblivious machines.

It is obvious that, for any time function $T(n)$, given a k-counter machine or a k-pushdown store machine, which operate in time $T(n)$, we can find a

* A preliminary version was presented at the 9th International Colloquium on Automata, Languages, and Programming held in Aarhus, Denmark, the third week of July 1982. This work is registered at the Mathematical Centre as IW 167/IW 197.

† In the autumn of 1983 the "Mathematical Centre" changes its name to "Centre for Mathematics and Computer Science."

0019-9958/82 $2.00

time equivalent k-tape Turing machine (k-TM). However, such an "obvious" Turing machine simulation will, apart from using k storage tapes, also use an amount of $\Theta(T(n))$ storage. In Pippenger and Fischer (1979) it was shown that, for the pushdown store, of which the contents cannot be appreciably compacted, the best we can do for on-line simulation by an oblivious Turing machine is 2 storage tapes, $\Theta(T(n) \log T(n))$ time and $\Theta(T(n))$ storage. For the multicounter machine, Fischer *et al.* (1968) demonstrated a linear time/logarithmic space simulation by a 1-tape Turing machine. In Corollary 2 of Schnorr (1976) it was shown how to simulate a $T(n)$ time-, $S(n)$ storage-bounded multitape Turing machine on-line by an oblivious 2-tape Turing machine in time $O(T(n) \log S(n))$ and storage $O(S(n))$. Combining the compacting of counts of Fischer *et al.* (1968) and the method of Schnorr (1976) we achieve the best previously known on-line simulation of a k-counter machine by an oblivious Turing machine: 2 tapes, $O(T(n) \log \log T(n))$ running time and $O(\log T(n))$ storage. It is somewhat surprising to see that we can restrict a Turing machine for on-line simulation of a k-counter machine to 1 storage tape, logarithmic storage, oblivious head movements and still retain a linear running time.

In Section 2 this result is derived and connected with a linear cost combinational network for doing the same job. This network processes the inputs in sequence and may incur a time delay of $\Theta(\log n)$ between processing an input and producing the corresponding output (followed by the processing of the next input). Since we would like to obtain a constant data rate, i.e., a constant time delay between processing the ith input at the ith input port and producing the ith output at the ith output port, $1 \leqslant i \leqslant n$, we show in Section 3 how to real-time simulate n steps of a multicounter machine by an oblivious $\log^* n$-head tape unit and use this to obtain a linear cost combinational network with such a fast response time. (Recall that $\log^* n$ is the number of consecutive iterations of taking the logarithm to get a number less than or equal to 1, when we start from n.) It is not our purpose here to introduce an odd machine model with a variable number of access pointers. One should rather think of it as an expedient intermediate step to derive the desired result for fixed n. Subsequently, we note that cyclic networks or VLSI circuits (where the length of the wires adds to the cost) can real-time simulate a multicounter machine in logarithmic (area) cost. (Recall that VLSI stands for very large scale integrated circuits.) In Section 5 we analyse the cost of simulating a multicounter machine on-line by a logarithmic cost RAM. This turns out to be $O(n)$ time and $O(\log n)$ space by the oblivious version, which is optimal, also for nonoblivious RAMs. For the relevant definitions of multicounter machines (Fischer and Rosenberg, 1968; Fischer *et al.*, 1968), multitape Turing machines (e.g. Rosenberg, 1967), combinational logic networks (Pippenger and Fischer, 1979), real-time and linear time on-line simulation (Pippenger and Fischer,

1979) and oblivious computations (Pippenger and Fischer, 1979; Schnorr, 1976) we direct the reader to these references.

2. Linear-Time On-Line Simulation by Oblivious One-Head Tape Units with an Application to Combinational Logic Networks

We first point out one of the salient features of the problem of simulating k-CMs on-line by efficient oblivious Turing machines. Suppose we can simulate some abstract storage device S on-line by an efficient oblivious Turing machine M. Then we can also simulate a collection of k such devices $S_1, S_2, \ldots, S_k$, interacting through a common finite control, by dividing all tapes of M into k tracks, each of which is a duplicate of the corresponding former tape. Now the same head movements do the same job on k collections of tracks as formerly on the tapes of M, so the time and storage complexity of the extended M are the same as those of the original. While the problem of, say, simulating a k-counter machine in linear time by a k'-tape Turing machine, for $k' < k$, stems precisely from the fact that k' is less than k, the problem of simulating a k-counter machine by a k'-tape oblivious Turing machine in linear time is the same problem as that of simulating a 1-counter machine in linear time by a k'-tape oblivious Turing machine. Hence, for a proof of feasibility it suffices to look for the simulation of 1 counter only. (For a proof of infeasibility we would have the advantage of knowing that the head movements are fixed, and are the same for all input streams. Besides, we could assume that we needed to simulate an arbitrary, albeit fixed, number of counters.) Fischer *et al.* (1968) observed that a 1-TM can simulate a k-CM on-line in linear time. Their simulation uses $O(\log n)$ storage, for n steps by the k-CM, which is clearly optimal. It is a priori by no means obvious that an oblivious multitape TM can simulate but *one* counter in linear time. We shall show that the cited result can be extended to hold for oblivious Turing machines as well.

In our investigation we noted that head reversals are not necessary to maintain counters. We did not succeed in getting the idea below to work in an oblivious environment, and include it here as a curiosity, possibly folklore, item. Suppose we want to simulate a k-CM C with counts $x_1, x_2, \ldots, x_k$ represented by the variables n_1 through n_k. The number of simulated steps of C is contained in the variable n. For $i = 1, 2, \ldots, k$, if count x_i is incremented by $\delta \in \{-1, 0, +1\}$, then

$$n_i \leftarrow n_i + 2 \quad \text{for} \quad \delta = +1;$$

$$n_i \leftarrow n_i + 1 \quad \text{for} \quad \delta = 0;$$

$$n_i \leftarrow n_i \quad \text{for} \quad \delta = -1.$$

Let us, for $i = 1, 2,..., k$, denote the current count on the ith counter of C by $\hat{x}_i$.

THEOREM 1. *For $i = 1, 2,..., k$, we have $\hat{x}_i = 0$ iff $n_i = n$.*

Proof. Let n be the number of steps performed by C. Let p_i be the number of $+1$'s, r_i be the number of 0's, and q_i be the number of -1's, added to the ith counter, $1 \leqslant i \leqslant k$, during these n steps. So $p_i + q_i + r_i = n$ for all i, $1 \leqslant i \leqslant k$. By definition we have $n_i = 2p_i + r_i$. Suppose $n_i = n$. Then it follows that $p_i = q_i$ and therefore $p_i - q_i = \hat{x}_i = 0$. Conversely, let $\hat{x}_i = p_i - q_i = 0$. Then $p_i = q_i$ and $n_i = p_i + q_i + r_i = n$. ∎

Hence we obtain:

COROLLARY. *A one-way k-CM C can be simulated in real-time by a $(k + 2)$-head* one-way non-writing finite automaton *F of which the heads can detect coincidence. Hence, four heads without head reversals suffice to accept all recursively enumerable sets.*

(Hint: 1 head reads the input from left to right, 1 head keeps the count of n by its distance to the origin, and the remaining k heads so keep the counts n_1 through n_k. It was shown by Minsky (1961) that 2-CMs can accept all recursively enumerable sets. We assume that the tape is unbounded, whatever the input may be.) After this digression we show

THEOREM 2. *If C is a k-counter machine, then we can find an oblivious 1-tape Turing machine M that simulates C on-line in time $O(n)$ and storage $O(\log n)$ for n steps by C.*

Following Pippenger and Fischer (1979), we note that in Theorem 2, "machine" can be replaced by "transducer" and the proof below will still hold.

Proof. It shall follow from the method used, and is also more generally the case for simulation by oblivious Turing machines (cf. above), that if the theorem holds for 1-CMs, then it also holds for k-CMs, $k \geqslant 1$. Let C be a 1-CM. The simulating oblivious 1-TM M will have one storage tape divided into 3 channels, called the *n-channel*, the *y-channel*, and the *z-channel*. The storage tape is one-way infinite to the right, with the leftmost square called the 0th or *start* square. At the start of the computation the storage tape head is positioned on the start square. If, in the current step of C its count c is modified to $c + \delta$, $\delta \in \{-1, 0, +1\}$, then

$$\begin{aligned} \delta = +1 \quad &\Rightarrow \quad n \leftarrow n + 1; y \leftarrow y + 1; z \leftarrow z, \\ \delta = 0 \quad &\Rightarrow \quad n \leftarrow n + 1; y \leftarrow y; z \leftarrow z, \\ \delta = -1 \quad &\Rightarrow \quad n \leftarrow n + 1; y \leftarrow y; z \leftarrow z + 1, \end{aligned}$$

where n is the count contained on the n-channel, y is the count contained on the y-channel and z is the count contained on the z-channel. Hence, always

$$c = y - z, \tag{1}$$

and

$$y + z \leqslant n. \tag{2}$$

The count n on the n-channel is recorded in the usual binary notation, with the low order digit in the start square and the high order digit on the right, see Fig. 1. For any input sequence, we call the interval of steps by the simulating oblivious machine M, used to simulate the ith step of the simulated counter machine C, the ith *cycle*.

At the start of the cycle simulating the ith step of C, for $i = p2^j > 0$ and p is odd, squares 0 through $j-1$ on the n-channel contain 1's and square j contains a 0. So in this cycle, M's head, starting from square 0, travels right to square j and deposits a 1 there. It turns all 1's on squares 0 through $j-1$ into 0's during this pass. The head then returns to square 0. This maintenance of the count n completely fixes M's head movements, so M is oblivious.

The representation of y and z is in a redundant binary notation. If y is denoted by $y_0 y_1 \cdots y_i$, with y_j in square j of the y-channel $(0 \leqslant j \leqslant i)$, then $y_j \in \{0, 1, 2\}$ and $y = \sum_{j=0}^{i} y_j 2^j$. Similarly for the count z. So the representation of $y[z]$ over $\{0, 1, 2\}$ is not unique. We also use a distinguished symbol called *blank*, denoted by $-$, to fill the squares of the infinite nonsignificant portion of a tape channel. At the start of the simulation, all squares of the channels of M's tape contain blanks. Finally, the head covers 2 squares on the tape, and shifts 1 square in 1 step of M. The head is like a *window* covering 2 tapesquares and its *position* is the position of the left square it covers. So it has a look-ahead of 1. The left symbol under scan is said to be the symbol in the *left window* and the right symbol under scan is the symbol in the *right window*.

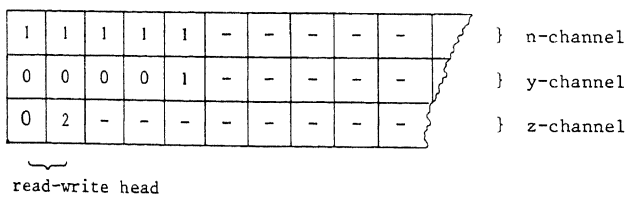

FIG. 1. The configuration on M's tape after it has simulated 31 steps of C, consisting of, consecutively, 16 *add* 1's, 11 *add* 0's, and 4 *add* -1's. The head has returned to the start position.

We now explain the operation of M. The intuitive idea behind a 2 in square j of the $y|z|$-channel is an as yet unprocessed carry from the jth to $(j+1)$th position of the binary representation of $y|z|$. During the left-to-right sweeps of the head, governed by the moves indicated for the updating of n, the machine maintains invariants (1) and (2). During the corresponding right-to-left sweeps, back to the start square, M maintains also invariant (3). We maintain (with allowance for the borderline case $j=0$),

$$\forall j \geqslant 0[(y_j > 0 \Rightarrow z_{j-1}, z_j, z_{j+1} \in \{0, -\}) \\ \& (z_j > 0 \Rightarrow y_{j-1}, y_j, y_{j+1} \in \{0, -\})], \tag{3a}$$

and

$$\forall j \geqslant 0[(y_j = - \Rightarrow y_{j-1} \in \{-, 1, 2\}) \\ \& (z_j = - \Rightarrow z_{j-1} \in \{-, 1, 2\})]. \tag{3b}$$

The latter condition gets rid of nonsignificant leading 0's. (The reader will notice that the notation for a count c, according to the invariants (1)–(3) as above, basically consists of a *symmetric redundant binary* representation using the digits $-2, -1, 0, 1, 2$, by invariant (3b) without leading nonsignificant 0's. The positive digits of this representation are stored on the y-channel, the negative digits are stored, in positive form, on the z-channel. As usual a digit d in position j contributes $d2^j$ to the total count. Invariant (3a) means that a proper positive digit and proper negative digit cannot be adjacent, there has to be a 0 in between.)

The validity of the simulation is now ensured if we can show the following assertions to hold at the end of M's cycle to simulate the ith step of C, for all $i \geqslant 0$.

(a) For all i, $i \geqslant 1$, the simulator M can always add 1 to either channel y or z in the cycle simulating step i of C.

(b) M can maintain invariants (1)–(3) to hold at the end of each simulation cycle.

(c) The fact that (1)–(3) hold at the end of the ith simulation cycle of M, for all $i \geqslant 0$, ensures that the count of C is 0 subsequent to C's ith step iff both the y-channel and z-channel contain blanks on all squares subsequent to the completion by M of the ith cycle.

CLAIM 1. Assertion (a) holds at the start of each simulation cycle.

Proof of claim. In the process of simulating the ith step of C, M takes care of (a) during its left-to-right sweep, by propagating all unprocessed carries on squares $0, 1, \ldots, j$ on both the y-channel and z-channel to the right, leaving only 0's or 1's (and possibly blanks) on squares $0, 1, \ldots, j$ and

deposing a digit d, $0 \leqslant d \leqslant 2$ (or possibly a blank), on square $j+1$ of the channel concerned, for $i = p2^j$ and p is odd. The input is processed by, at the start of each cycle, incrementing the digit in the start square of the appropriate channel by the proper amount. This leads to at most an unprocessed carry in the start square. Assuming that M has adopted this strategy, we prove the claim by induction on the number of steps of C, equivalently, the number of simulation cycles of M.

Clearly, the claim holds at the start of the first cycle. Suppose the claim holds for simulation cycles $1, 2, \ldots, i-1$, then it also holds for the ith cycle, since

Case 1 $(i = 2^j)$. At the start of this cycle the count on channel $y|z|$ can be at most $2^j - 1$. At the end of the right sweep the head covers square j. Since the count, on either channel, now has reached at most 2^j, it suffices to put a 0 or 1 in square j. The relevant carries can always be propagated, since by the inductive assumption the encoding scheme sufficed up to the ith cycle, and so the maximum count on squares 0 through h $(0 \leqslant h < j)$ on a channel is less than 2^{h+2} because

$$\sum_{j=0}^{h} 2.2^j = 2(2^{h+1} - 1) = 2^{h+2} - 2.$$

Case 2 $(i = p2^j$, with $p > 1$ and p odd). The square on the channels scanned in the left window of the head, at the rightmost position of this sweep, is square j. The last time square j was scanned in the left window was 2^j cycles ago. The cycle concerned was i', with $i' = (p-1)\,2^j$. During cycle i' also square $j+1$ was scanned in the left window, since $i' = ((p-1)/2)\,2^{j+1}$. Hence, under the assumption that the scheme of simulating steps $1, 2, \ldots, i-1$ of C by M was carried out correctly, square $j+1$ contains no 2 at the start of cycle i, since it was left with a blank, 0 or 1 in cycle i' and has not been visited since. The maximum count left, at the end of the i'th cycle, in squares $0, 1, \ldots, j$ of either channel, was $2^{j+1} - 1$. Since then, 2^j cycles have passed, and therefore the count to be represented, by squares $0, 1, \ldots, j+1$ of either channel, cannot exceed

$$2^{j+1} - 1 + 2^j + 2^{j+1} = 2.2^{j+1} + 2^j - 1,$$

which certainly can be taken care of by a 2 in square $j+1$ (covered by the right window of the head in cycle i) and 1's in squares 0 through $j-1$. By the same reasoning as in Case 1 all necessary intermediate carries, left on squares 0 through j, by cycles $i'+1$ through $i-1$, can be propagated right during the current left-to-right sweep, leaving squares 0 through j with blanks, 0's or 1's, and square $j+1$ with $d \in \{-, 0, 1, 2\}$, when the head returns to the origin, for both the y-channel and z-channel.

Hence a left-to-right sweep can always update the y and z count appropriately, under the assumed strategy of M, during its oblivious head movements governed by the updating of the n-count. ∎

CLAIM 2. Assertion (b) holds at the start of each simulation cycle.

Proof of Claim. As we saw in the proof of Claim 1, assertion (a) is implemented during the left-to-right sweeps. During the right-to-left sweeps assertion (b) is implemented. Clearly, assertion (b) holds at the start of the 1th cycle. During its right-to-left sweeps, M subtracts in each step the 2-digit numbers covered on the y- and z-channel from each other, leaving the scanned positions on at least one channel containing only 0's. M also changes leading 0's on either channel into blanks during its right-to-left sweeps. Suppose the claim holds at the start of simulation cycles 1, 2,..., i. We show that then it also holds at the start of simulation cycle $i+1$. It is obvious that M's strategy outlined above maintains invariants (1) and (2). It is left to show that it also maintains invariant (3).

Case 1 ($i = 2^j$). The count on the y-channel (z-channel) can be at most 2^j. Therefore, at the start of the ith right-to-left sweep, the head covers the most significant digits on either channel, or is to the right of them, while on its right-to-left sweep it encounters only blanks, 0's or 1's. Moving left, it subtracts the lesser number covered from the greater (or equal) number on the other channel, at each step, meanwhile leaving blanks instead of leading 0's on either channel. The following situations can arise. (The symbol $\varnothing$ stands for either a "0" or a "–".) If $b + 2c = e + 2f$,

$$\cdots \begin{matrix} a & b & c \\ d & e & f \end{matrix} \cdots \vdash_M \cdots \begin{matrix} a & \varnothing & \varnothing \\ d & \varnothing & \varnothing \end{matrix} \cdots . \tag{i}$$

If $b + 2c > e + 2f$,

$$\cdots \begin{matrix} a & b & c \\ d & e & f \end{matrix} \cdots \vdash_M \cdots \begin{matrix} a & b' & c' \\ d & \varnothing & \varnothing \end{matrix} \cdots , \tag{ii}$$

where $b' + 2c' = b + 2c - e - 2f$. If $b + 2c < e + 2f$,

$$\cdots \begin{matrix} a & b & c \\ d & e & f \end{matrix} \cdots \vdash_M \cdots \begin{matrix} a & \varnothing & \varnothing \\ d & e' & f' \end{matrix} \cdots , \tag{iii}$$

where $e' + 2f' = e + 2f - b - 2c$. Since $i = 2^j$, at the outset of the right-to-left sweep the head has blanks in its right window, since the maximal position containing nonblank digits is square j. Hence there will be no problem turning leading 0's, created in the right-to-left cleaning, into blanks during the travel to the low order square. Thus we maintain invariant (3b).

Suppose that invariant (3a) is not satisfied after the right-to-left sweep. Say, $y_h > 0$ and not all of z_{h-1}, z_h, and z_{h+1} are 0 or blank, for some $h \geqslant 2$. Let $z_{h+1} > 0$. Then, according to (i)–(iii), the move

$$\cdots \begin{matrix} a & b & c \\ d & e & f \end{matrix} \cdots \vdash_M \cdots \begin{matrix} a & \varnothing & \varnothing \\ d & e' & f' \end{matrix} \cdots,$$

with $f' = z_{h+1} > 0$, must have been the move of the right-to-left sweep leaving the $(h+1)$th square, and for all values of a, d, e' the next move must be

$$\cdots \begin{matrix} a & \varnothing & \varnothing \\ d & e' & f' \end{matrix} \cdots \vdash_M \cdots \begin{matrix} a' & \varnothing & \varnothing \\ d' & e'' & f' \end{matrix} \cdots,$$

which contradicts $y_h > 0$. Let $z_h > 0$, or let $z_{h-1} > 0$. This also leads to a contradiction with $y_h > 0$, as we leave for the reader to check. For $h \in \{0, 1\}$ the argument proceeds similarly, with allowance for the borderline case.

Case 2 ($i = p2^j$, with $p > 1$ and p odd). At the start of the right-to-left sweep, the square in the left window on either channel is square j. At the start of this cycle, invariant (3) is satisfied for the complete tape, by the inductive assumption. Therefore, at the start of the right-to-left sweep it is satisfied for all squares $h \geqslant j+3$, since at most square $j+1$ can have been changed by the head in this cycle. Moreover, either square $j+2$ on the y-channel, or square $j+2$ on the z-channel contains a 0 or blank. So at the start of the right-to-left sweep we can assume that the situation is

$$\cdots \begin{matrix} y_{j-1} & y_j & y_{j+1} & y_{j+2} \\ z_{j-1} & z_j & z_{j+1} & \varnothing \end{matrix} \cdots.$$

The last time square $j+1$ was covered was in cycle i', which was 2^j cycles ago. According to the inductive assumption, condition (3) was satisifed at the end of that cycle. Moreover, since $i' = ((p-1)/2)\,2^{j+1}$, we have according to the proof of Claim 1 that squares 0 through $j+1$ contained only 0's, 1's, or blanks at the end of that cycle. Assume that at the end of cycle i' it holds $y_{j+1} > 0$. Then, also at the end of that cycle, $z_j, z_{j+1}, z_{j+2} \in \{0, -\}$. Hence, the maximum count on squares 0 through $j+2$ on the z-channel, in that cycle, was $2^j - 1$. So in the current cycle i, the maximum count on these squares of the z-channel becomes at most $2.2^j - 1 = 2^{j+1} - 1$. Consequently, at the start of the current right-to-left sweep $z_{j+1}, z_{j+2} \in \{0, -\}$. Recapitulating, if at the start of the current right-to-left sweep $y_{j+1} > 0$ then $z_{j+2}, z_{j+1} \in \{0, -\}$, and, similarly, if $z_{j+1} > 0$

then $y_{j+2}, y_{j+1} \in \{0, -\}$. Hence, at the start of this right-to-left sweep, condition (3) is fulfilled for all squares $h \geqslant j+2$, and if $z_{j+1}|y_{j+1}| > 0$ then also $y_{j+1}|z_{j+1}| \in \{0, -\}$, with all leading 0's turned into blanks up to, and including, square $j+1$. So Case 2 reduces to Case 1, except in case $y_{j+1}|z_{j+1}| > 0$, when the head starts its right-to-left sweep in the ith cycle, and the subtraction of $z_{j+1}z_j|y_{j+1}y_j|$ from $y_{j+1}y_j|z_{j+1}z_j|$ creates new leading 0's, which have to be turned into blanks. This difficulty, however, is easily circumvented either by marking the most significant digits on the y- and z-channels, or by giving the head an extra look ahead. This proves the claim. ∎

CLAIM 3. Assertion (c) holds at the start of each simulation cycle.

Proof of Claim. That a square on a channel can contain a blank only if all squares right of it, on that channel, contain blanks, and that the representations of y and z have no leading 0's, at the start of each simulation cycle, is a consequence of the proof of Claim 2. That $y - z = c$ at the conclusion of the ith simulation cycle of M, where c is the count of C after i steps, follows because in the left-to-right sweep we add the correct amount to a channel according to Claim 1, and in the right-to-left sweep we substract equal amounts from either channel. It remains to show that as a consequence of the maintenance of condition (3) assertion (c) holds under these conditions.

Suppose that, at the end of the ith simulation cycle of M, not both the y- and z-channel contain but blanks and that, by way of contradiction, $y - z = 0$. Then there is one channel, say y, which has a leading digit in position $j, j > 0$, while the digits in the positions j and $j-1$ on the z-channel are blank. So the count represented by y is greater or equal to 2^j while the count on z is smaller or equal to $2\sum_{i=0}^{j-2} 2^i = 2^j - 2$. So $y - z \geqslant 2$ which contradicts the assumption. (For $j = 0$ we obtain $y - z \geqslant 1$.)

It remains to show that if $c \neq 0$, then not both channels y and z can contain only blanks. Since always, at the start of a cycle, $c = y - z$ holds, if $c \neq 0$ then $y \neq z$; so in that case at least one of the y-channel and z-channel must contain a count unequal to zero. Hence there must be a square which contains a digit $d > 0$ on one of these channels. ∎

By Claims 1–3 the on-line simulation of C by M is correct as outlined. It is easy to see that the simulation uses $O(\log n)$ storage for simulating n steps by C. We now estimate the time required for simulating n steps by C. In the ith simulation cycle M needs to travel to square j, for $i = p2^j$ and p is odd. Therefore, M needs $2j$ steps for this cycle. For cycle i with $i = p2^j$ and p is even, i.e., i is even, M needs 1 step. Hence, for 2^{h+1} steps by C, the simulator M needs all in all,

$$
\begin{aligned}
T(2^{h+1}) &= \sum_{j=1}^{h} (2^{h-j} \cdot 2j) + 2^h \\
&= 2^{h+1} \sum_{j=1}^{h} (j \cdot 2^{-j}) + 2^h \\
&< 2^{h+1} \sum_{j=1}^{\infty} (j \cdot 2^{-j}) + 2^h \\
&\leqslant 2 \cdot 2^{h+1} + 2^h \\
&= 5 \cdot 2^h.
\end{aligned}
$$

Now, given n, choose $h = \lfloor \log n \rfloor$ so that $2^h \leqslant n < 2^{h+1}$. Then $T(n) \leqslant T(2^{h+1}) \leqslant 5 \cdot 2^h \leqslant 5n$. Since the movement of M's head has nothing to do with the actual counts y and z, but only with the number of steps passed since the start of C, we observe that a k-CM can be simulated on-line by an oblivious 1-tape TM M_k, which is just like M, but equipped with y_i- and z_i-channels, $1 \leqslant i \leqslant k$, and therefore with a total of $2k + 1$ channels. Just like M, the new M_k uses $\Theta(\log n)$ storage and $T(n) \leqslant 5n$ steps to simulate n steps of C_k, the simulated k-CM, which proves the theorem. The covering of 2 or 3 tape squares by the head of M can be simulated easily by cutting out 1 or 2 squares of the storage tape and buffering it in the finite control. The swapping to and fro, from tape to buffer, according to the storage head movement, is easily handled by a slightly larger finite control. This is similar to the way to achieve the speed-up of Hartmanis and Stearns (1965). ∎

It is well known that oblivious Turing machine computations correspond to those of *combinational logic networks*; cf. Pippenger and Fischer (1979) or Schnorr (1976). The networks we consider are acyclic interconnections of *gates* by means of *wires* that carry signals. It will be assumed that there are finitely many different types of gates available and that these form a "universal" basis, so that any input–output function can be implemented by a suitable network. Each type of a gate has a *cost*, which is a positive real number, say 1 for each. The *cost* of a network is the sum of the costs of its gates. The method used above can be used to construct a combinational logic network that implements the first n steps of the computation by a k-CM. Such a network will have n inputs carrying suitable encodings of the symbols read from the input terminal and n outputs carrying encodings of the symbols written to the output terminal, where we assume, for technical reasons, that the k-CM is a transducer. If the input- and output-alphabets have more than two symbols, the inputs and outputs of the network will be "cables" of wires carrying binary signals. Using standard techniques, as in the references above, it is easy to show, by imitation of the oblivious Turing machine constructed in the proof of Theorem 2, that:

COROLLARY. *If C is a k-CM transducer, then we can construct a combinational logic network implementing n steps of C with cost $O(kn)$.*

3. REAL-TIME SIMULATION BY AN OBLIVIOUS $\log^* n$-HEAD TAPE UNIT AND A CORRESPONDING COMBINATIONAL LOGIC NETWORK

In the simulations of the previous section we may incur a time delay of $\Theta(\log n)$ between the processing of an input and the production of the corresponding output. For the combinational logic network with n input ports and n output ports this is interpreted as follows. The $(i+1)$th input port is enabled by a signal from the ith output port. Between this enabling and the production of the $(i+1)$th output $\Theta(\log n)$ time may pass. Note that we can only process the $(i+1)$th input after the ith output is produced, since the set of zero counts at step i influences the translation of the jth input to incrementing/decrementing the various counters, for all $j > i$. To eliminate the unbounded time delay we construct as an intermediate step, for each n, a real-time simulation by an oblivious $\log^* n$-head tape unit. While this does not solve the classic problem of simulating an arbitrary multicounter machine in real-time by a Turing machine with a fixed number of tapes (Fischer and Rosenberg, 1968; Fischer *et al.*, 1968), it turns out that, with respect to the resulting combinational logic network, the construction yields as good a result as could be obtained from a real-time simulation of an arbitrary multicounter machine by an oblivious Turing machine with a fixed number of tapes. In the sequel we call a combinational network with $\Theta(1)$ time delay, between enabling the ith input port and the production of the ith output, a *constant data rate* network.

For the $\log^* n$-head simulation we use basically that of the previous section with the tape divided into $\log^* n$ blocks of increasing sizes, each with a resident head. The size of the 0th block is $s(0) = x$, for some constant x. The size of block 1 is $s(1) = 2^{x-1}$ and the size of block i, $i > 1$, is $s(i) = 2^{s(i-1)}$. Since we need $\Theta(\log n)$ length tape to simulate n steps, we need less than $\log^* n$ blocks. The 0th block is maintained in the finite control and, assuming the blocks are marked off, all heads can travel around on local information alone. Only the head on block 1 needs to be connected with the finite control to exchange information regarding the counts. See Fig. 2.

The encoding of the current counts uses y- and z-channels to store the redundant binary integer representations, and "–" denotes the distinguished blank symbol, as before. Each head covers four squares, like the two-square *window* in the previous section, and is said to be positioned on the leftmost square it covers. Each head, on information which is put in the first square

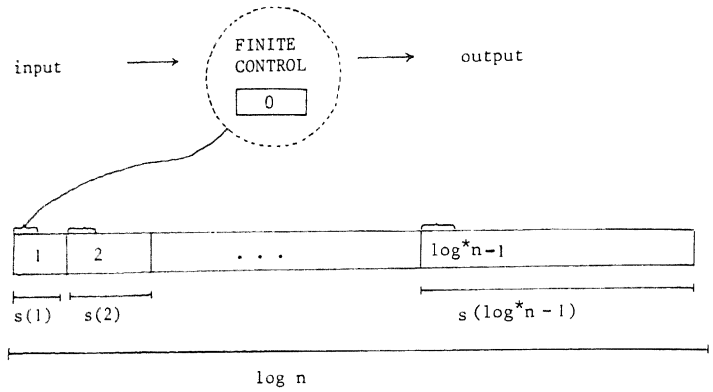

FIGURE 2

of its block, by the head on the previous block, makes a sweep from left-to-right over its block until it scans the end cell and then back from right-to-left until it scans the first cell. There it waits until the next sweep is due. Hence such a complete sweep over block i, by the resident head, takes $2s(i)$ steps. We maintain three invariants.

At all times $t \geqslant 0$ holds:

$$y + z \leqslant t, \tag{4}$$

$$y - z = \text{current count}, \tag{5}$$

and for all positions j on blocks 0 through $\log^* n$, denoted by $j \in J$,

$$\underset{j \in J}{\forall} j[(y_j > 0 \Rightarrow z_{j-1}, z_j, z_{j+1} \in \{0, -\}) \\ \& (z_j > 0 \Rightarrow y_{j-1}, y_j, y_{j+1} \in \{0, -\})], \tag{6a}$$

and

$$\underset{j \in J}{\forall} j[(y_j = - \Leftrightarrow z_j = -) \& \neg(y_j = z_j = 0 \& y_{j+1} = z_{j+1} = -)]. \tag{6b}$$

(For $j = 0$ the obvious allowances are made.) The movements of the heads are governed by the count on the n-channel. Here this count may contain 2's representing unprocessed carries. This does not occur on the segment of n maintained on block 0, which is incremented by 1 in each step. When that count reaches 0 again (modulo 2^x steps), a carry is sent to the head on block 1 which then resides on the first square. Upon receiving a carry from block 0, the head on block 1 makes a full sweep over block 1, processing the carry, and returns to the first square. Since this takes $2s(1) = 2^x$ steps, it is

duly in position to receive the next carry. When the segment of the n count in block 1 reaches 0 again, modulo $2^{s(1)}$ sweeps, at the right extreme of the last of these sweeps a carry is propagated to the first square of block 2. This carry, in its turn, instigates a sweep of the resident head over block 2. For each $i \geqslant 0$, we associate with block i the length $T(i)$ of the shortest *interval* of steps in between the production of two consecutive nonzero carries from block i to block $i+1$. In general, each cycle of $2^{s(i)}$ sweeps over block i produces a carry to the first square of block $i+1$, starting a sweep by the resident head. Since such a sweep takes $2s(i+1)$ steps, and a carry to block $i+1$ is produced at the end of intervals of $T(i)$ steps, with $T(i) \geqslant 2s(i)\,2^{s(i)}$, the head on block $i+1$ is in position to start its sweep upon receiving the carry if

$$2s(i+1) \leqslant 2s(i)\,2^{s(i)}, \tag{7}$$

for all $i \geqslant 1$. Block 0 is instantly updated, and therefore we need

$$2s(1) \leqslant 2^{s(0)}. \tag{8}$$

Since inequalities (7) and (8) are satisfied by the chosen block sizes, each propagated carry to a block is processed immediately. Having fixed the oblivious head movements, by starting a sweep over block $i+1$ each time a carry arrives from block i on the n channel, it remains to prove that the present invariants (4)–(6) can be maintained at all times during the real-time simulation. (Before proceeding, we remark that it is not necessary to assume that the blocks are delimited on the tape initially. Using four extra counters we can, as soon as we have the size of block i on one of then, determine $s(i+1)$ before the first sweep over block $i+1$ is due. Determining the size of block 1 by the finite control, we can *bootstrap* the simulation of these four counters in the main simulation itself, which thereby will be able to simulate an arbitrary number of counters, and so succesively determine the blocks as they are needed. However, for the present objective of eventually producing a combinational logic network, there is no advantage in amplifying on this construction.) We have to show:

(a) Each block can always receive incoming carries on the first square of its $y-|z-|$ channel, and, in particular, block 0 receiving the inputs never overflows. That is, (4) and (5) are maintained at all times.

(b) Invariant (6) holds at all times.

From (a) and (b) it follows, by the same reasoning as in the last section, that the current count $y-z=0$ iff both $y=z=0$ iff both y- and z-channel currently contain blanks only iff both y- and z-channel contain a blank in the first position. The finite control, containing block 0, therefore knows instantly when the count is zero.

CLAIM 1. Statement (a) can be maintained.

Proof Sketch. By induction on the consecutive blocks i.

Base case. A sweep over block 1 takes $2s(1) = 2^{s(0)}$ steps. Since a channel y or z can accommodate a count of $2(2^{s(0)} - 1)$ on block 0, it follows that, subsequent to the propagation of a carry (signifying a count of $2^{s(0)}$) to block 1, block 0 contains at most $2^{s(0)} - 1$ on either channel. In the next $2^{s(0)} - 1$ steps the count may rise to $2(2^{s(0)} - 1)$, but at the $2^{s(0)}$th step a new carry is propagated to block 1, restoring a count of at most $2^{s(0)} - 1$.

Induction. During its left-to-right sweeps, the head on block i, $i > 0$, processes a 2 deposited in the first square of the y, z-channels, of that block, by propagating it as far as possible on the left two squares in the window. So a 2 in the first square of a channel of block i may increment the contents of the first square of that channel on block $i + 1$ by 1.

Assume that the first square of a channel on block j, $1 \leqslant j \leqslant i$, is incremented by at most 1 in between the starts of two consecutive sweeps over that block. Identifying 0's and blanks, and considering only one channel, let block i contain $000 \cdots 0$ or $10 \cdots 0$ at the start of the t_1th sweep. By assumption, if block i contains $211 \cdots 1$ at the start of the t_2th sweep, $t_2 > t_1$, then $t_2 - t_1 \geqslant 2^{s(i)} - 1$. So sweep t_2 causes an increment of 1 on the first square of block $i + 1$, by propagating the concerned 2 right, leaving just 0's. Also by the assumption, at the start of the $(t_2 + 1)$th sweep, block i contains $00 \cdots 0$ or $10 \cdots 0$ again. Since block i initially contains blanks only, and $t_2 - t_1 + 1 \geqslant 2^{s(i)}$, while a sweep over block $i + 1$ takes less time than $2^{s(i)}$ sweeps over block i, the assumption is shown to hold for block $i + 1$ too, thus supplying the induction step. The assumption holds for block 1 by the base case. So no channel on a block i, $i > 0$, ever contains a local count of more than $2^{s(i)} + 1$ which, together with the base case, proves the claim. ∎

CLAIM 2. Statement (b) can be maintained.

Proof Sketch. Contrary to the simulation in the previous section, we preserve invariant (6) while going from left-to-right, on a block, in propagating a carry. Going from right-to-left nothing is changed, so invariant (6) will hold at all times. We do so by subtracting the 3 bit pieces of the y- and z-count, in the left three windows, while going from left-to-right. If a nonzero digit replaces a 0 or a blank on a channel, this is required to be in the middle window of the three, and the three positions covered on the other channel are replaced by 0's (or blanks). This still allows us to propagate a 2 as far as the central position of the three windows, so to the first square on the next block at the right extreme of the sweep. From the proof of Claim 1 we have seen that a carry to the first square of the next block was sufficient.

The rightmost (fourth) square covered by the head serves to detect adjacent blanks, so as to return created leading 0's to blanks immediately. Such a look-ahead suffices for this purpose, due to the fact that, since invariant (6) holds and 2's can occur only on the first square of a block, and in the window, only *one* new nonsignificant leading 0 can be created, per channel, in one sweep on the rightmost nonblank block. ∎

Hence we have

THEOREM 3. *We can simulate the first n steps of a multicounter machine by an oblivious* $\log^* n$*-head tape unit in real-time and logarithmic space. (Similarly we can directly construct an oblivious* $\log^* n$*-tape Turing machine for the same job.)*

Just as we argued in the previous section, we can construct a corresponding combinational logic network. Since only the squares which are being rewritten need to be represented by logic components, and the time to make a sweep over block $i+1$ is $2s(i+1)$, while there is only one such sweep in each consecutive interval of $T(i)$ steps, with $T(i) \geqslant 2s(i)\,2^{s(i)} = 2s(i)\,s(i+1)$, the cost of this network is reduced from the expected $\Theta(n \log^* n)$ by not representing squares covered by a head which does no rewriting.

THEOREM 4. *We can implement the first n steps of a k-counter machine on an* $O(kn)$ *cost combinational logic network with a constant data rate.*

Proof. The network has a constant data rate, i.e., a time interval $O(1)$ between enabling the ith input port by the $(i-1)$th output and producing the ith output, $1 \leqslant i \leqslant n$, since it is derived from a real-time simulation. Each piece of logic circuitry, representing four squares covered by a head which is moving, has cost $c(k)$ linear proportional to the number k of counters simulated. (So $c(k)$ is independent of the number of steps n.) The state of the finite control (containing block 0) is represented by cost $d(k)$ pieces of logic connected to the input ports; $d(k)$ is a linear function of k too. In each consecutive interval of $T(i) \geqslant 2s(i)\,2^{s(i)}$ steps, the head on block $i+1$ is active for only $2^{s(i)+1}$ steps. Hence such a head is active for only $O(n/s(i))$ steps out of n, for all i, $1 \leqslant i < \log^* n$. Summing this for all blocks i, and adding the cost for the blocks 0 connected to the input ports, we obtain a total cost $C(k, n)$,

$$C(k,n) = n(c(k) + d(k)) + \sum_{i=1}^{\log^* n - 1} \frac{nc(k)}{s(i)} = O(kn). \quad \blacksquare$$

For the knowledgeable reader we add that the $\log^* n$-head tape unit can be viewed as an *iterative array* with a limited amount of oblivious local activity.

4. Simulation by Cyclic Networks and VLSI

When we are not restricted to acyclic logic networks, but are allowed cyclic logic networks, or work in the framework of the VLSI model of computation recently advanced in Mead and Conway (1980), it is not difficult to see

THEOREM 5. *If C is a k-CM transducer, then we can construct*:

(i) *a cyclic logic network, simulating n steps of C in real-time, with cost $O(k \log n)$;*

(ii) *a VLSI circuit, simulating n steps of C in real-time, with area $O(k \log n)$.*

Proof. We prove (ii), which clearly implies (i). The VLSI circuit realizing the claimed behaviour could look as indicated in Fig. 3. Each row of logics blocks stores one count in ordinary binary notation, with the low digit contained in the left block. Each block stores two bits: one for the binary digit of the count, and one to indicate whether the count digit contained is the most significant bit of that count. Carries are propagated along the top wire of each row, borrows along the bottom wire. The middle wires of each row transport information concerning the most significant bit in that row. Each block contains the necessary logic to process and transmit correctly carries, borrows, and information concerning the most significant bit. The finite-control-logic rectangle processes the input signals and the information from the leftmost block of each row, viz. whether they contain a most significant bit 0 of the corresponding count, to issue carries or borrows to the leftmost block of each row and to compute the output signal. We leave it to the reader to confirm that, subsequent to receiving the input signal, the corresponding output signal can be computed in time $O(\log k)$, which corresponds to the bit length of an input signal for driving k counters. Hence the VLSI circuit simulates the k-CM in real-time. Since the area occupied by the wires, emanating from each block, can be kept to the same size as the

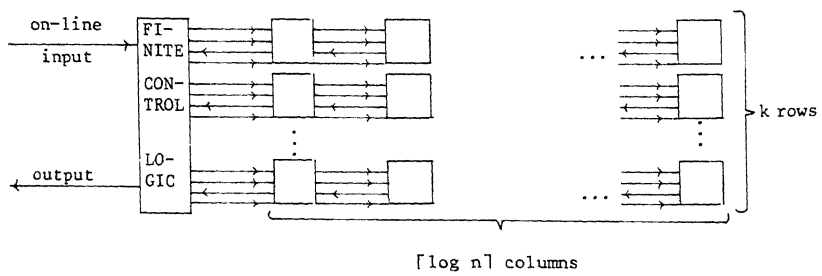

FIG. 3. VLSI circuit simulating k-counter machine.

area occupied by the block itself, the rows of blocks, together with the wires, take all together $O(k \log n)$ area. The finite control logic structure contains some trees of depth $\log k$, so its area can be kept to $O(k \log k)$. Under the assumption that $k \in O(n)$ this yields the required result. ∎

To fit a long thin rectangle in a square, as often is necessary to implement the structure on chip, we can fold it without increasing the surface area significantly. Note that the structure contains no long wires, and that it does not have to be overal synchronized: local synchronization is all we need. Hence it is a practical design. In fact, the design is a particular type of VLSI circuit known as *linear systolic array*; cf. Mead and Conway (1980).

5. Simulation by RAMs

For simulation with a uniform cost RAM it is clear that we can simulate a multicounter machine on-line with constant delay and constant storage. Constant delay is the RAM analogue for real-time, i.e., if $T(n)$ is the time for simulating n steps by the multicounter, then the RAM simulates on-line with constant delay if $T(n+1) - T(n) < c$ for some constant c and all n. It is easy to see, that a logarithmic cost RAM cannot simulate a real-time counter machine on-line with constant delay, since such a RAM can only address registers of bounded index and bounded contents.

At first glance it seems that we can do no better than $O(n \log n)$ time for simulation of a counter machine by a logarithmic cost RAM. If we simulate with a tally mark in each register, we have to use indirect addressing to maintain the top of the counter, requiring $O(n \log n)$ time and $O(n)$ storage to simulate n steps. Using a binary count we need only k registers for a k-counter machine, but need again $O(n \log n)$ time and $O(\log n)$ storage. Define an *oblivious* RAM as one in which the sequence of executed instructions, as well as the sequence of accessed storage locations, is a function of time alone. Due to the usual restrictions of the arithmetic operations of RAMs to $+$ and $-$, as well as to the needed translation of input commands with respect to the set of currently zero counters into the basic counter instructions, we need to augment the RAM with some constant-bit-length Boolean/arithmetic instructions in order not to be artificially precluded from obtaining the following result by imitation of the simulation in Section 2. (If we do not add these extra operations Theorem 6 might only hold for nonoblivious RAMs by purely irrelevant definitional reasons.) Since we view the RAM as an abstract storage device performing a transduction, we also assume it is connected to the input and an output terminal, and dispense with the usual "accept" instruction. Using the simulation in Section 2 we obtain

THEOREM 6. *We can simulate a k-counter machine on-line by an oblivious logarithmic cost* RAM *in* $O(kn)$ *time and* $O(k \log n)$ *storage.*

Proof. Implement the simulation of Section 2 on a RAM, storing the head position of the 1-tape Turing machine in register 1 and the jth square contents in register $j+1$. Then the sequence of executed instructions in the RAM program, and the sequence of accessed registers, can be made a function of time alone. So the RAM is oblivious. The time for simulating sweeps of length j on the RAM is

$$O\left(k \sum_{i=2}^{j+1} \log i\right) = O(kj \log j).$$

So if $T(2^{h+1})$ is the time needed to execute the first 2^{h+1} steps of the multicounter we obtain

$$T(2^{h+1}) \in O\left(\sum_{j=1}^{h} (k2^{h-j} j \log j) + k2^h\right) = O(k2^{h+1}).$$

So $T(n) \in O(kn)$ and the storage used is $O(k \log n)$. ∎

This simulation is optimal in both space and time, even for nonoblivious RAMs.

6. FINAL REMARKS

Comparing our solution of the linear time simulation of a k-CM with the nonoblivious one of Fischer *et al.* (1968), the reader will notice that our average time complexity is the same as the worst case time complexity there. So in actual fact, that earlier solution runs often faster than the one presented here. Fischer and Rosenberg (1968) showed that the Origin Cossing Problem: "report when all k counts simultaneously reach 0" admits a real-time one-tape Turing machine solution. Contrary to the former linear time simulation, the latter method seems to contain inherently nonoblivious features, preventing us from turning it into an oblivious version. It has been a classic question, cf. the cited references, whether or not the Axis Crossing Problem: "report when one out of k counters reaches 0" or more generally "on-line simulate a k-counter machine" can be done in real-time by a (nonoblivious) k'-tape Turing machine for $k' < k$. A reasonable approach may seem to show that, anyway, a real-time simulation of multicounter machines by *oblivious* one-head tape units is out of the question. In the event, intuition is wrong. We have noticed, cf. Section 2, that if we restrict the simulation device to its oblivious counterpart we have the advantage that

if 1 counter is simulatable, then k counters can be simulated in just the same way. This key observation has led us in the meantime, by augmenting the ideas presented here with an involved tape manipulation technique, to a real-time simulation of multicounter machines by *oblivious* one-head tape units, thus solving the above problem with a considerable margin, Vitányi (1982). Although superficially it would seem that this farther reaching result obviates the present ones we like to point out that:

(a) The present results are far simpler to derive and will suffice for many applications, as will some of the distinctive techniques.

(b) To derive the linear cost, constant data rate, combinational logic network the present route by way of a $\log^* n$-head tape unit suffices.

(c) The RAM simulation result seems difficult to derive, if at all, from the simulation in Vitányi (1982) without regressing to the simulation given here.

REFERENCES

FISCHER, M. J. AND ROSENBERG, A. L. (1968), Real-time solutions of the origin-crossing problem, *Math. Systems Theory* **2**, 257–264.

FISCHER, P. C., MEYER, A. R., AND ROSENBERG, A. L. (1968), Counter machines and counter languages, *Math. Systems Theory* **2**, 265–283.

HARTMANIS, J., AND STEARNS, R. E. (1965), On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* **117**, 285–306.

MEAD, C. A., AND CONWAY, L. A. (1980), "*Introduction to* VLSI *Systems*," Addison–Wesley, New York.

MINSKY, M. (1961), Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines, *Ann. of Math.* **74**, 437–455.

PIPPENGER, N., AND FISCHER, M. J. (1979), Relations among complexity measures, *J. Assoc. Comput. Mach.* **26**, 361–384.

ROSENBERG, A. L. (1967), Real-time definable languages, *J. Assoc. Comput. Mach.* **14**, 645–662.

SCHNORR, C. P. (1976), The network complexity and Turing machine complexity of finite functions, *Acta Inform.* **7**, 95–107.

VITÁNYI, P. M. B. (1982), Real-time simulation of multicounters by oblivious one-tape Turing machines, *in* "Proceedings 14th Annual ACM Symposium on Theory of Computing," pp. 27–36, Assoc. Comput. Mach., New York. (Final version: "An optimal simulation of counter machines," *SIAM J. Comput.*, to appear.)