

**stichting  
mathematisch  
centrum**



---

AFDELING INFORMATICA  
(DEPARTMENT OF COMPUTER SCIENCE)

IW 167/81

MEI

P.M.B. VITÁNYI

EFFICIENT IMPLEMENTATIONS OF MULTICOUNTER MACHINES  
ON OBLIVIOUS TURING MACHINES, ACYCLIC LOGIC NETWORKS,  
AND VLSI

Preprint

---

**kruislaan 413 1098 SJ amsterdam**

---

1980 Mathematics subject classification: 68C40, 68C25, 68C10  
ACM Computing Reviews-categories: 5.23, 5.25, 5.26.

Efficient implementations of multicounter machines on oblivious Turing machines, acyclic logic networks, and VLSI.\*)

by

Paul M.B. Vitányi

#### ABSTRACT

A 1-tape oblivious Turing machine can simulate a  $k$ -counter machine online in linear time and logarithmic space. This leads to a linear cost combinational logic network implementing  $n$  steps of a  $k$ -counter machine. In the VLSI model of computation we can simulate  $n$  steps of a  $k$ -counter machine in real-time on area  $O(k \log n)$ . A  $k$ -counter machine can be simulated in real-time by a (nonoblivious) machine without head reversals. Some results are stated about oblivious  $k$ -counter languages.

KEY WORDS & PHRASES: *Counter machines, combinational logic networks, oblivious Turing machines, linear time simulation, complexity measures, VLSI*

---

\*) This report will be submitted for publication elsewhere.



## 1. INTRODUCTION

In many computations it is necessary to maintain several counters such that, at all times, an instant signal indicates which subset of the counts is zero. Keeping  $k$  counts in tally notation, where a count is incremented (decremented) by at most 1 in each step, is formalized in the notion of a  $k$ -counter machine.  $k$ -counter machines have been studied extensively, because of their numerous connections with both theoretical issues and practical applications, in the computer science literature of the last decade. 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 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. 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 timing function  $T(n)$ , given a  $k$ -counter machine, or a  $k$ -pushdown store machine, which operate in time  $T(n)$ , we can find a time equivalent  $k$ -tape Turing machine. However, such a Turing machine will, apart from using  $k$  tapes, also use  $O(T(n))$  storage. In [7] it was shown that for the pushdown store, of which the contents can not 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 counter machine, [2] demonstrated a linear time-logarithmic space simulation by a 1-tape Turing machine. [9, Corollary 2] showed how to simulate on-line a  $T(n)$  time-,  $S(n)$  storage-bounded multitape Turing machine 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 in [2] and the method of [9] 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 we give some definitions and preliminaries, and, by way of illustration, discuss the notion of oblivious counter machines. In section 3 we derive the main result and connect this with a linear cost (acyclic) combinational logic network for simulating a  $k$ -counter machine. Subsequently we show that it is straightforward to simulate a  $k$ -counter machine in real-time by a logarithmic cost cyclic logic network or by a VLSI in logarithmic area cost. In section 4 we indicate some open problems, like real-time simulation of counter machine (problems) by (oblivious) Turing machines, and pinpoint the shift in emphasis, in the problem to be solved, occasioned by the transition to oblivious machines: whereas in the simulation by nonoblivious machines the main difficulty lies in the simulation of more counters by less tapes, in the simulation by oblivious machines the difficulty lies only in the obliviousness of the simulating device. We also point out that the implementations for counter machines as described are optimal, up to a constant multiplicative factor, for the claimed simultaneous resource bounds.

## 2. DEFINITIONS AND PRELIMINARIES

A *one-way on-line  $k$ -counter machine* ( $k$ -CM) consists of a finite-state control,  $k$  *counters* each capable of containing any integer, and an input terminal. The states of the finite control are partitioned into *polling* and *autonomous* states. At the start of the computation the CM is in a designated initial state and all counters are set to *zero*. A *step* in a CM computation is uniquely determined by the state of the control unit, by the symbol scanned at the input terminal if the state is a polling state, and by the set of counters which contain zero. The action at that step consists of independently altering the contents of each counter by adding 0, +1 or -1, and changing the state of the control unit. If the new state entered by the control unit is a polling state, the machine also outputs a 0 or a 1, as part of the step leading up to the new state, indicating "rejection of the sequence of input symbols so far" or "acceptance of the sequence of input symbols so far". Let  $\Sigma$  be the set of input symbols. The *storage* required by a CM in processing

$w \in \Sigma^*$  is the sum of the maximum absolute values of the contents of each counter in the course of the  $t$  steps of the computation. The time used in processing  $w \in \Sigma^*$  is the number  $t$  of steps used. Let  $T$  and  $S$  be monotone increasing functions. A counter machine operates in time  $T$  [in storage  $S$ ] if, for all  $n \geq 0$ , when CM is started on an input of length  $n$ , the CM polls the input terminal for the  $(n+1)$ -th symbol, at a step  $t \leq T(n)$  and requires space not exceeding  $S(n)$  in its computation up to step  $t$ . A language  $L$  over  $\Sigma$  is *recognized* by a counter machine if CM accepts every word in  $L$  over  $\Sigma$  and rejects every word in  $\Sigma^* - L$ .  $L$  is recognizable in *time*  $T(n)$  [in *storage*  $S(n)$ ] if there is a CM recognizing  $L$  which operates in time  $T$  [in storage  $S$ ]. We single out the cases  $c'n \leq T(n) \leq cn$ , for some constants  $c', c$ , as *linear time*, and  $T(n) = n$  as *real-time* (i.e., every reachable state is a polling state). The significance of these time restrictions (especially the latter) stems from its importance in syntactic analysis and real-time control. More formal definitions can be found in [2], together with many results on  $k$ -CMs. The model is, for the time-restricted case, not sensitive to mild changes in convention, except that going from one-way to two-way makes a difference but for the real-time case. Although CMs might seem pretty restricted machines, which is indeed the case for 1-CMs, in [4] it was shown that already 2-CMs (without time / space restrictions) can simulate a universal Turing machine!

An *on-line  $k$ -tape Turing machine* ( $k$ -TM) is, and works, just like a  $k$ -CM except that the storage structure does not consist of  $k$  counters but of  $k$  tapes, each with one read-write two-way head. For formal definitions and results on on-line  $k$ -tape Turing machines, see [8,10]. Pertinent to the discussion here is that  $k$ -tape Turing machines with time bounds are insensitive to mild changes in definition; in particular, if  $T(n) = O(n)$  and  $T(i) - T(i-1) \leq c$ , for some constant  $c$  and all  $i$ , for some Turing machine  $M$ , then we can find a Turing machine  $M'$  which recognizes the same set and operates in real-time.  $M$  is said to operate with *constant delay*, and the result follows from the constant speed-up as in [3].

We shall say that two machines *simulate* each other, if, when they are started and subsequently receive the same sequence of input symbols from their input terminals, they produce the same sequence of symbols as output. Clearly, simulation is an equivalence relation. We shall say that the simulation is *on-line*, if the polling of the input terminal and the

outputting of symbols occurs in the same order (but not necessarily at the same steps) for both machines. This is also an equivalence relation. We say that one machine that simulates another does so in time  $T(n)$  if for every  $n$ , the one machine outputs at least as many output symbols in the first  $T(n)$  steps, as the other machine outputs during its first  $n$  steps.

Finally, we explain the notion of an oblivious machine. A Turing machine is *oblivious*, if the polling for input, writing of output and the movements of the storage heads are fixed functions at time, independent of the inputs to the machine. (Cf. [5,6,9,11,12]). One may think of the head movements, the polling for input and the writing of output, as being governed by a second autonomous machine which has no input terminal. The same definition can obviously be applied to machines like RAMs, requiring that the sequence of instructions followed and the sequence of storage locations accessed each be independent of the input. Many problems seem inherently oblivious; the usual algorithms for computing  $+$ ,  $-$ ,  $*$ , and  $/$  or a table look-up can easily be programmed obliviously. Not so a problem like binary search. One reason to study oblivious Turing machine computations is that they are easily translated in combinational logic networks. Another is that, when we restrict ourselves to this class of computations, we are often able to derive lower bounds on time complexity, or time-space trade-offs, of computations. See also [7,9].

Note that the notion of oblivious counter machines does not make too much sense, since if the motion of the counters is governed by the set of zero counts only, in effect such a machine can only accept the intersection of a regular set (due to its finite control) and a counter language over a one-letter alphabet (the length set checked by the counters). Hence we have:

THEOREM 1. *The languages recognized by oblivious 1-CM's are precisely the regular languages.*

PROOF. The one counter either increases indefinitely or cycles through a count which is bounded by a constant.  $\square$

However,

THEOREM 2. (i) *The set of languages recognized by an oblivious real-time*



*2-CM contains strictly content sensitive languages. (ii) The set of languages accepted by an oblivious 2-CM contains nonrecursive languages.*

PROOF. In [4] it was shown that 2-CMs can simulate universal Turing machines. Hence (ii) follows by letting a 2-CM accept a nonrecursive unary language. (i) follows from (ii) by taking the same machine, change all states to polling states, and letting it expect a 1 when it polls as before, and a 0 when it polls in a state which used to be autonomous. Since the set accepted yields a nonrecursive language under erasing of 0's, the set must be non-context free. It is content sensitive since it is accepted in real-time.  $\square$

In general, for each language  $L$  accepted by an oblivious  $k$ -CM with time bound  $T(n)$ ,

$$L = h^{-1}(L') \cap R$$

where  $R$  is a regular set,  $L'$  is a one-letter language accepted by an oblivious  $k$ -CM, and  $h$  is a homomorphism which maps  $a \rightarrow 1$  for all letters  $a$  in the alphabet of  $L$ .

After this digression in oblivious counter machines, we turn to some salient features of the problem of simulating  $k$ -CM's 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  $S_1, S_2, \dots, S_k$ , 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 real-time by a  $k'$ -tape Turing machine,  $k' < k$ , 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 real-time is the same problem as that of simulating a 1-counter machine in real-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 have the advantage of knowing that the head movements are fixed, and are the same for all input streams. Besides, we can assume that we needed to simulate an arbitrary, albeit fixed, number of counters.

### 3. ON-LINE SIMULATION OF k-COUNTER MACHINES BY OBLIVIOUS TURING MACHINES, IMPLEMENTATIONS ON ACYCLIC LOGIC NETWORKS, AND SIMULATION BY VLSI CIRCUITS

In [2] the somewhat surprising result was shown, that a 1-TM can simulate a k-CM on-line in linear time. This 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 multi tape TM can simulate but one counter in linear time. We shall show that the result of [2] can be extended to hold for oblivious Turing machines.

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  $x_1, x_2, \dots, x_k$  represented by the variables  $n_1$  through  $n_k$ . The number of simulated steps of  $C$  is contained in the variable  $n$ . For  $i = 1, 2, \dots, k$  if count  $x_i$  is incremented by  $\delta \in \{-1, 0, +1\}$  then

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

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

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

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

**LEMMA 3.** For  $i = 1, 2, \dots, k$ ,  $\hat{x}_i = 0$  iff  $n_i = n$ .

**PROOF.** Let  $n$  be the number of steps performed by  $C$ ,  $p_i$  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  $i$ -th counter,  $1 \leq i \leq k$ , during these  $n$  steps. Hence  $p_i + q_i + r_i = n$  for

all  $i$ ,  $1 \leq i \leq k$ . 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  $n_i = p_i + q_i + r_i = n$ .  $\square$

Hence we obtain:

COROLLARY. A  $k$ -CM  $C$  can be simulated in real-time by a  $(k+2)$ -head one-way nonwriting finite automaton  $F$  of which the heads can detect coincidence. Hence, 4 heads 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 in [4] that 2-CMs can accept all recursively enumerable sets.)

After this digression we now turn to an extension of the method used in [2], and show how it can be made to work in an oblivious environment, in order to obtain:

THEOREM 4. 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 [7], we note that in the above theorem "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. last paragraph of section 2 and section 4), that if the Theorem holds for 1-CM's then it also holds for  $k$ -CM's,  $k \geq 1$ . 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. If, in the current step of  $C$  its count  $c$  is modified to  $c+\delta$ ,  $\delta \in \{-1, 0, +1\}$ , then:

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

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 (1)  $c = y - z$ , and (2)  $y + z \leq n$ . The count  $n$  on the  $n$ -channel is recorded in the usual binary notation, with the low order digit on the start square and the high order digit on the right, see Figure 1. At the start of the cycle simulating the  $i$ -th step of  $C$ ,  $i = p \cdot 2^j$  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  $y_0 y_1 \dots y_i$ ,  $y_j$  in square  $j$  of the  $y$ -channel, then  $y_j \in \{0, 1, 2\}$ ,  $0 \leq j \leq i$ , 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. Finally, the head covers 2 squares on the tape, and shifts 1 square in 1 step of  $M$ , like a mask covering 2 tapesquares. So it has a look-ahead of 1. See Figure 1.

1	1	1	1	1	-	-	-	-	-		} n-channel
0	0	0	0	1	-	-	-	-	-		} y-channel
1	2	-	-	-	-	-	-	-	-		} z-channel


  
 read-write head

**Figure 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 5 "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  $j$ -th to  $(j+1)$ -th position of the binary representation of  $y[z]$ . During the left-to-right sweeps of its head, governed by the moves indicated for the updating of  $n$ ,  $M$  maintains invariants (1) and (2). During the corresponding right-to-left sweeps back to the start square,  $M$  maintains also invariant (3):

if  $y_j [z_j] > 0$  is the contents of squares  $j$  on the  $y[z]$  channel then  $z_{j-1}, z_j, z_{j+1} [y_{j-1}, y_j, y_{j+1}]$  are 0 or blank. Moreover, every square right of a blank square, on that channel, contains blanks and no square containing a 0 has a blank right neighbor in that channel. This latter condition gets rid of leading 0's.

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  $i$ -th step of  $C$ ,  $i \geq 1$ .

- (a) For all  $i$ ,  $i \geq 1$ ,  $M$  can always add 1 to either channel  $y$  or  $z$  in the cycle simulating step  $i$  of  $C$ .
- (b)  $M$  can maintain invariants (1), (2) and (3) to hold at the end of each simulation cycle.
- (c) The fact that (1), (2) and (3) hold at the end of the  $i$ -th simulation cycle of  $M$  ensures that the count of  $C$  is 0 subsequent to  $C$ 's  $i$ -th step iff both the  $y$ -channel and  $z$ -channel contain blanks on all squares subsequent to the completion by  $M$  of simulating  $C$ 's  $i$ -th step.

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

PROOF OF CLAIM. In the process of simulating the  $i$ -th step of  $C$ ,  $M$  takes care of (a) during its left-to-right sweep, by propagating all unprocessed carries on squares  $0, 1, \dots, j$  on both the  $y$ -channel and  $z$ -channel to the right, leaving 0's or 1's on squares  $0, 1, \dots, j$  and depositing a digit  $d$ ,  $0 \leq d \leq 2$ , on square  $j+1$  of the channel concerned, for  $i = p \cdot 2^j$  and  $p$  is odd. Assuming that  $M$  has adopted this strategy, we prove the claim by induction on the number of steps of  $C$ , equivalently, number of simulation cycles of  $M$ .

Clearly, the claim holds at the start of the first cycle. Suppose the claim holds for simulation cycles  $1, 2, \dots, i-1$ , then it also holds for the  $i$ -th 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 the maximum count on squares 0 through  $h$  on a channel is less than  $2^{h+2}$  because

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

Case 2.  $i = p \cdot 2^j$ ,  $p > 1$  and  $p$  odd. The square on the channels scanned by the left part of the head, in its rightmost position of this sweep, is square  $j$ . The last time square  $j$  was scanned by the left part of the head was  $2^j$  cycles ago, and at that cycle  $i'$ ,  $i' = (p-1)2^j$ , also square  $j+1$  was scanned by the left part of the head, since  $i' = ((p-1)/2)2^{j+1}$ . Hence, under the assumption that the scheme of simulating step  $1, 2, \dots, i-1$  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  $0, 1, \dots, j$  of either channel, was  $2^{j+1}-1$ . Since then,  $2^j$  cycles have passed, and therefore the count to be represented, by squares  $0, 1, \dots, j+1$  of either channel, cannot exceed

$$2^{j+1}-1 + 2^j + 2^{j+1} = 2 \cdot 2^{j+1} + 2^j - 1,$$

which certainly can be taken care of by a 2 in square  $j+1$  (covered by the right part 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  $d \in \{\text{blank}, 0, 1, 2\}$ , 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.  $\square \square$

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 1-th cycle. During its right-to-left sweeps, at each step  $M$  subtracts the 2-digit numbers

covered on the y- and z-channel from each other, leaving the covered 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  $1, 2, \dots, i$ . We show that it then 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  $2^j$ . Hence the head covers the most significant digits on either channel, while on its right-to-left sweep it only encounters blanks, 0's or 1's. Moving left, it subtracts the lesser number covered by the head 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:

$$(i) \quad \begin{array}{ccccc} \dots & a & b & c & \dots \\ & d & e & f & \\ & & \underbrace{\phantom{e f}}_{\leftarrow} & & \end{array} \quad \vdash_M \quad \begin{array}{ccccc} \dots & a & \emptyset & \emptyset & \dots \\ & d & \emptyset & \emptyset & \\ & & \underbrace{\phantom{d \emptyset}}_{\leftarrow} & & \end{array}$$

if  $bc = ef$ ;

$$(ii) \quad \begin{array}{ccccc} \dots & a & b & c & \dots \\ & d & e & f & \\ & & \underbrace{\phantom{e f}}_{\leftarrow} & & \end{array} \quad \vdash_M \quad \begin{array}{ccccc} \dots & a & b' & c' & \dots \\ & d & \emptyset & \emptyset & \\ & & \underbrace{\phantom{d \emptyset}}_{\leftarrow} & & \end{array}$$

if  $cb > fe$ , where  $c'b' = cb - fe$ ;

$$(iii) \quad \begin{array}{ccccc} \dots & a & b & c & \dots \\ & d & e & f & \\ & & \underbrace{\phantom{e f}}_{\leftarrow} & & \end{array} \quad \vdash_M \quad \begin{array}{ccccc} \dots & a & \emptyset & \emptyset & \dots \\ & d & e' & f' & \\ & & \underbrace{\phantom{d e'}}_{\leftarrow} & & \end{array}$$

if  $cb < fe$ , where  $f'e' = fe - cb$ . For the sake of the picture we have denoted both 0's and blanks by  $\emptyset$ . Since  $i = 2^j$ , at the outset of the right-to-left sweep the head has blanks under 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.

Suppose, condition (3) is not fulfilled 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,  $h \geq 2$ . Let  $z_{h+1} > 0$ . Then, since

$$\begin{array}{ccc} \dots & a & b & c & \dots \\ & d & e & f & \\ & & \underbrace{\phantom{e f}}_{\leftarrow} & & \end{array} \quad \vdash_M \quad \begin{array}{ccc} \dots & a & \emptyset & \emptyset & \dots \\ & d & e' & f' & \\ & & \underbrace{\phantom{e' f'}}_{\leftarrow} & & \end{array}$$

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

$$\begin{array}{ccc} \dots & a & \emptyset & \emptyset \\ & d & e' & f' \\ & & \underbrace{\phantom{e' f'}}_{\leftarrow} & \end{array} \quad \vdash_M \quad \begin{array}{ccc} \dots & a' & \emptyset & \emptyset & \dots \\ & d' & e'' & f' & \\ & & \underbrace{\phantom{e'' f'}}_{\leftarrow} & \end{array} ,$$

this 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.  $i = p \cdot 2^j$ ,  $p > 1$  and  $p$  is odd. At the start of the right-to-left sweep, the square covered by the left side of the head on either channel is square  $j$ . At the start of this cycle, condition (3) is satisfied for the complete tape, according to the induction assumption, so at the start of the right-to-left sweep it is satisfied for all squares  $h \geq j+3$ , since at most square  $j+1$  can be changed by the right part of the head. 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



$$\begin{array}{ccccccc}
 \dots & y_{j-1} & y_j & y_{j+1} & y_{j+2} & \dots \\
 & z_{j-1} & z_j & z_{j+1} & \emptyset & \\
 & & \underbrace{\phantom{z_j}} & & & \\
 & & \leftarrow & & & 
 \end{array}$$

where, for the sake of the argument, we identify 0 and blank. The last time square  $j+1$  was covered was at cycle  $i'$ ,  $2^j$  cycles ago. According to the induction assumption, condition (3) was satisfied at the end of that cycle and, moreover, since  $i' = ((p-1)/2)2^{j+1}$ , according to the proof of claim 1, 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  $i'$ ,  $y_{j+1}[z_{j+1}] > 0$ . Then, also at the end of that cycle,  $z_j, z_{j+1}, z_{j+2} [y_j, y_{j+1}, y_{j+2}] \in \{0, \text{blank}\}$ . Hence, the maximum count on squares 0 through  $j+2$  on the  $z$ -[ $y$ -] channel of that cycle was  $2^j - 1$ . So in the current cycle  $i$ , the maximum count on these squares of the  $z$ -[ $y$ -] channel becomes at most  $2 \cdot 2^j - 1 = 2^{j+1} - 1$ . Therefore, at the start of the current right-to-left sweep  $z_{j+1}, z_{j+2} [y_{j+1}, y_{j+2}] \in \{0, \text{blank}\}$ . So if at the start of the current right-to-left sweep  $z_{j+1} > 0$  then  $y_{j+2}, y_{j+1} \in \{0, \text{blank}\}$  and if  $y_{j+1} > 0$  then  $z_{j+2}, z_{j+1} \in \{0, \text{blank}\}$ . Hence, at the start of this right-to-left sweep, condition (3) is fulfilled for all squares  $h \geq j+2$ , and if  $z_{j+1}[y_{j+1}] > 0$  then also  $y_{j+1}[z_{j+1}] \in \{0, \text{blank}\}$ , with all leading 0's turned into blanks up to, and including, square  $j+1$ . So case 2 reduces to case 1, except for the case that  $y_{j+1}[z_{j+1}] > 0$  when the head starts its right-to-left sweep at the  $i$ -th 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 by either marking the most significant digits on the  $y$ - and  $z$ -channels, or by giving the head an extra look ahead.

This proves the claim.  $\square \square$

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

PROOF OF CLAIM. That a square on a channel can only contain a blank 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  $i$ -th 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 subtract 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  $i$ -th simulation cycle of  $M$ , not both the  $y$ - and the  $z$ -channel contain but blanks and that, by way of contradiction,  $y-z = 0$ . Assuming for the sake of the argument that negatively indexed squares contain blanks, and identifying 0's and blanks, we can represent  $y-z$  by

$$y-z = \sum_{j=-\infty}^{\infty} x_j 2^j,$$

with  $x_j = y_j - z_j$  for all  $j$ . It is a consequence of condition (3) that not both  $y_i$  and  $z_j$  are unequal to 0, and neither can  $x_j$  and  $x_{j+1}$  have an opposite sign. So if  $z_j = 0$  then  $x_j = y_j$  and if  $y_j = 0$  then  $x_j = -z_j$ . Furthermore, if  $x_j \in \{-1, -2\}$  and  $x_{j'} \in \{1, 2\}$ ,  $j' \neq j$ , then  $|j-j'| > 1$  and there is an integer  $h$  in between  $j$  and  $j'$  such that  $x_h = 0$ . If some square on the  $y$ - or  $z$ -channel contains a nonblank symbol we can, because there are no leading 0's, assume that there is a square, say the  $h$ -th one, on this channel containing a digit  $d \in \{1, 2\}$ , and  $|x_h| = d$ . Since the sequence  $\dots, x_{-1}, x_0, x_1, \dots, x_i, x_{i+1}, \dots$  contains no consecutive elements which have an opposite sign, and  $x_{-1} = x_{i+1} = 0$ , there must be integers  $\ell, r$ ,  $0 \leq \ell \leq h \leq r \leq i$ , such that  $x_\ell$  through  $x_r$  are unequal to 0 and of the same sign while  $x_{\ell-1} = x_{r+1} = 0$ . So under the assumptions (for  $\ell \geq 1$ ):

$$(a) \quad \left| \sum_{j=\ell}^r x_j 2^j \right| = \left| \sum_{j=-\infty}^{\ell-2} x_j 2^j + \sum_{j=r+2}^{\infty} x_j 2^j \right|;$$

$$(b) \quad 2^\ell \leq \left| \sum_{j=\ell}^r x_j 2^j \right| \leq 2^{r+2} - 2^{\ell+1};$$

$$(c) \quad 0 \leq \left| \sum_{j=-\infty}^{\ell-2} x_j 2^j \right| \leq 2^\ell - 2;$$

$$(d) \quad \text{either } \left| \sum_{j=r+2}^{\infty} x_j 2^j \right| = 0$$

$$\text{or } \left| \sum_{j=r+2}^{\infty} x_j 2^j \right| \geq 2^{r+2}.$$

We have now obtained a contradiction: the lefthand side of equation (a) has a value in  $[2^\ell, 2^{r+2} - 2^{\ell+1}]$  while the righthand side has a value in  $[0, 2^\ell - 2] \cup [2^{r+2} - 2^{\ell+2}, \infty)$ . The only remaining case  $\ell=0$  is also easily found to lead to a contradiction. So we cannot have both a nonblank symbol in a square of either the y- or z-channel and  $y-z = 0$ .

It remains to show that if  $c \neq 0$  then not both channels y and z 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  $\neq 0$ . Hence there must be a square which contains a digit  $d > 0$  on one of these channels.  $\square \square$

By claims 1, 2 and 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 i-th simulation cycle M needs to travel to square j, for  $i = p \cdot 2^j$  and p is odd. Therefore, M needs  $2j$  steps for this cycle. For  $i = p \cdot 2^j$  and p is even, i.e., i is even, M needs 1 step. Hence, for simulating  $2^{h+1}$  steps by C, M needs all in all:

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

Now, given n, choose  $h = \lfloor \log n \rfloor$  so that  $2^h \leq n < 2^{h+1}$ . Then  $T(n) \leq T(2^{h+1}) \leq 5 \cdot 2^h \leq 5n$ .

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  $C$ , it is easy to see 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,  $1 \leq i \leq k$ , and therefore with a total of  $2k+1$  channels. Just like  $M$ ,  $M_k$  uses  $\Theta(\log n)$  storage and  $T(n) \leq 5n$  steps to simulate  $n$  steps of  $C_k$ , the simulated  $k$ -CM, which proves the Theorem.

The covering of 2 tape squares by the head of  $M$  can be simulated easily by cutting out 1 square 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 in the finite control, of which the size is blown up a bit. This is similar to the way to achieve the speed-up in [3].  $\square$

It is well-known that oblivious Turing machine computations correspond to those of *combinational logic networks* [7,9]. 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 gate has a *cost* and a *delay*, which are positive real numbers, say 1 for each. The *cost* of a network is the sum of the costs of its gates. The *depth* of a network is the maximum over all input-to-output paths of the sum of the delays of the gates of that path. The method used in the previous section can also 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 on 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, [7,9], it is easy to show, by imitation of the oblivious Turing machine constructed in the proof of Theorem 4, that:

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

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 [5], it is not difficult to see that:

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

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

PROOF. We prove (ii), and (ii) clearly implies (i). The VLSI circuit realizing the claimed behavior could look as follows:

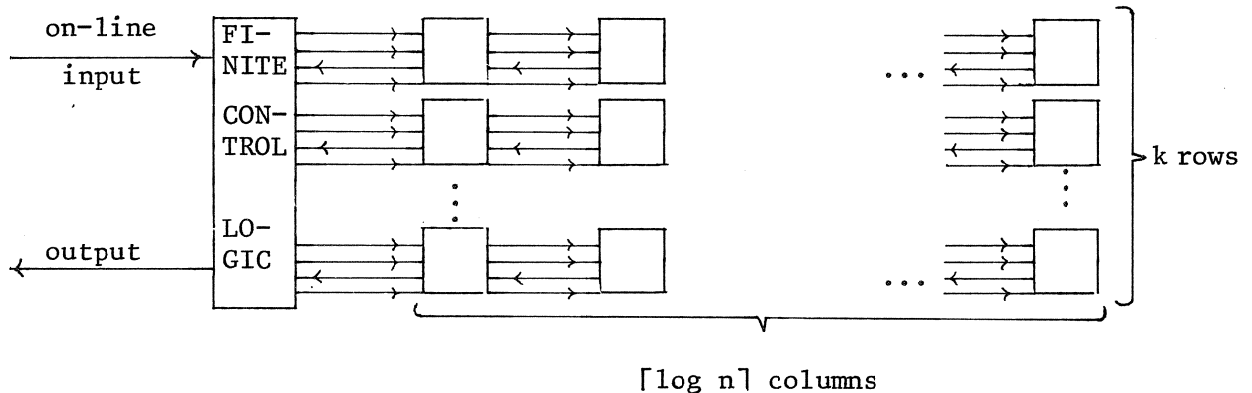


Figure 2. VLSI circuit simulating  $k$ -CM.

Each row stores a 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 first blocks of each row, whether they contain a most significant bit 0 of the corresponding count, to issue carries or borrows to the first 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 area occupied by the block itself, the blocks take  $O(k \log n)$  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.  $\square$

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 overall synchronized: local synchronization is all we need. Hence it is a practicable design.

#### 4. DISCUSSION AND SOME OPEN PROBLEMS

The counter machine as described is essentially an on-line device. Even if we take the off-line variant, where the input is read from an input tape delimited by markers, and the input read-write head is allowed a two-way motion governed by the state of the finite control, a zero count at any moment in the computation can influence all later counts. So any device simulating a counter machine, even the off-line variant, has no other option than to compute a representation of all intermediate counts in the counter machine computation. This shows two things:

- (i) any universal scheme for implementing a counter machine, both the on-line and the off-line variant, can take no shortcuts: it has to compute (a representation of) all intermediate counts of the counter machine;
- (ii) as a consequence of (i), all implementations of counter machines described are *optimal*, both for  $n$  steps by the on-line variant and  $n$  steps

by the off-line variant, up to a constant multiplicative factor, for the claimed simultaneous resource bounds of storage, time and area.

Comparing our solution of the linear-time simulation of a  $k$ -CM with the nonoblivious one in [2], the reader will notice that our average time complexity is the same as the worst-case time complexity in [2]. So in actual fact, the solution in [2] will run faster in most cases than the one presented here.

In [1] it was shown that the Origin Crossing Problem: "report when the  $k$  counters simultaneously reach zero" admits a real-time 1-tape Turing machine solution. Can this problem also be solved by a real-time oblivious  $k'$ -tape Turing machine for some  $k'$ ? For the related Axis Crossing Problem "report when one of the  $k$  counters reaches zero" and the related "simulate a  $k$ -CM in real-time" it is an open question whether this is doable by a  $k'$ -tape TM,  $k' < k$ , which is nonoblivious. It seems that such questions are naturally couched in terms of oblivious Turing machines. For consider the following observation:

THEOREM 6. *Let  $M$  be an oblivious  $k'$ -tape Turing machine which simulates a 1-CM in real time. Then we can find, for any  $k$ , an oblivious  $k'$ -tape Turing machine  $M_k$ , which simulates a  $k$ -CM in real-time.*

Hence the question of a real-time simulation of a  $k$ -CM, or a real-time solution to the Origin Crossing Problem or the Axis Crossing Problem, by an oblivious  $k'$ -TM,  $k' < k$ , reduces to the real-time simulation of a 1-CM by an oblivious multitape Turing machine. Apart from the fact that it is easier to prove things about oblivious machines, this reduction would seem to make it easier to obtain a (dis)affirmative answer to the question. Note that in the nonoblivious case the stress of the problem lies differently. It is easy to simulate a 1-CM by a 1-tape Turing machine in real-time: in  $O(n)$  storage the trivial way and, more difficultly, in  $O(\log n)$  storage as in the solution to the Origin Crossing Problem of [1]. Contrary to the linear time on-line simulation of [2], of a 1-CM by a 1-TM, the real-time simulation of [1] does not seem to extend to a simulation by an oblivious 1-TM. Here the problem lies exactly in the obliviousness of the simulating machine, and not in the fact that we must real-time maintain more counts on less tapes. Finally, in [12] we have used the result of section 3 to

show that the notion of *limited obliviousness*, [11,12], is not just disguised plain obliviousness, but indeed a new concept in between obliviousness and total nonobliviousness.

#### REFERENCES

- [1] FISCHER, M.J. & A.L. ROSENBERG, Real-time solutions of the origin-crossing problem, *Math. Systems Theory* 2 (1968), 257-264.
- [2] FISCHER, P.C., A.R. MEYER & A.L. ROSENBERG, Counter machines and Counter languages, *Math. Systems Theory* 2 (1968), 265-283.
- [3] HARTMANIS, J. & R.E. STEARNS, On the computational complexity of algorithms, *Trans. Amer. Math. Soc.* 117 (1965), 285-306.
- [4] MINSKY, M., Recursive unsolvability of Post's problem of tag and other topics in the theory of Turing machines, *Ann. of Math.* 74 (1961), 437-455.
- [5] MEAD, C.A. & L.A. CONWAY, Introduction to VLSI Systems. Addison-Wesley, New York, 1980.
- [6] PATERSON, M.S., M.J. FISCHER & A.R. MEYER, An improved overlap argument for on-line multiplication, SIAM-AMS Proceedings, Vol. 7, (Complexity of Computation) 1974, 97-112.
- [7] PIPPENGER, N. & M.J. FISCHER, Relations among complexity measures, *Journal ACM*, 26 (1979), 361-384.
- [8] ROSENBERG, A.L., Real-time definable languages, *Journal ACM* 14 (1967), 645-662.
- [9] SCHNORR, C.P., The network complexity and Turing machine complexity of finite functions, *Acta Informatica* 7, (1976), 95-107.
- [10] VITÁNYI, P.M.B., On the power of real-time Turing machines under varying specifications, extended abstract, *Lecture Notes in Computer Science* 85 (1980), 658-671, Springer Verlag, New York (Proc. 7th ICALP).
- [11] VITÁNYI, P.M.B., Relativized Obliviousness, *Lecture Notes in Computer Science* 88 (1980), 665-672, Springer Verlag, New York. (Proc. MFCS '80).



- [12] VITÁNYI, P.M.B., Complexity of limited oblivious computation, Tech.  
Rept. IW /81, Mathematisch Centrum, Amsterdam, 1981.

ONTVANGEN 17 JUNI 1981