AN OPTIMAL SIMULATION OF COUNTER MACHINES: THE ACM CASE

Paul M.B. Vitányi

AUTHOR'S ADDRESS:

Mathematisch Centrum
Kruislaan 413
1098SJ Amsterdam
The Netherlands

Proposed Running Head: THE ACM CASE

AN OPTIMAL SIMULATION OF COUNTER MACHINES: THE ACM CASE*

Paul M.B. Vitányi†

ABSTRACT

[SIAM J. on Computing, 14 (1985), 34-40]. This is a companion paper to ‘‘P.M.B. Vitanyi, An optimal simulation of counter machines, SIAM Journal on Computing, 14 (1985), 1-33, and the later ‘‘J. Seiferas and P.M.B. Vitanyi, Counting is easy, J. Assoc. Comp. Mach. 35 (1988), pp. 985-1000’’].

An Augmented Counter Machine (ACM) is a multicounter machine, with initially nonzero counters allowed, and the additional one-step instruction ‘‘set counter i to the value of counter j’’, for any pair of counters i and j. Each ACM can be real-time simulated by an oblivious one-head tape unit using the information- theoretical storage optimum.

Keywords & Phrases: counter machine, multicounter machine, augmented counter machine, real-time simulation by oblivious one-tape Turing machine, number representation, counting, coding.

Proposed Running Head: THE ACM CASE

---

* This work is registered at the Mathematical Centre as IW 225/83. It presupposes and continues [3].

† Author’s Address: Mathematisch Centrum, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. In the autumn of 1983 the *Mathematical Centre* changes its name to *Centre for Mathematics and Computer Science.*

1. Introduction

In the companion paper [3], a real-time implementation of multicounter machines on oblivious one-head tape units of optimal storage efficiency was exhibited. An *augmented counter machine (ACM)* is a multicounter machine, with each of its counters initialized to any integer, and with the additional one-step operation *''set counter i to the value of counter j''*, for any pair of counters i and j. Several one-step operations, other than the basic ones, can be synthesized on a multicounter machine by the use of concealed auxiliary counters (such as *''test equality of a pair of counters''*, for any such pair, by maintaining the differences on auxiliary counters). It is known that the above assignment among counters can not be so synthesized. A witness for this fact is the language L^*, with $L = \{ 0^p 1^m \mid p \geq m > 0 \}$. Thus, in real-time, ACM's are more powerful than multicounter machines [1]. The particular technique used in [3], to obtain an optimal simulation of counter machines, is well suited to extend that result to the more powerful ACM's. Consequently, we shall demonstrate:

THEOREM. *Each augmented counter machine can be real-time simulated by an oblivious one-head tape unit using the information-theoretical minimum in storage space. (Viz., for each $t \geq 0$ and $n \geq 1$, during the processing of the $(t+1)$-th through $(t+n)$-th input command, of the simulated ACM, the storage head of the simulating oblivious one-head tape unit accesses but $O(\log n)$ distinct tape squares.)*

In [3] the analog of the Theorem was derived for the weaker multicounter machines. The next section, containing the demonstration of the above Theorem, continues and presupposes that paper.

Outline of the simulation. The simulation consists of the oblivious one-head tape unit constructed in [3], equipped with some additional features. The $k+1$-track one-head tape unit M of [3], capable of real-time simulation of a k-counter machine, has one tag track, which does not concern us here, and k count tracks containing the momentary representations of the k stored integers, one per track. M would trivially be capable of real-time simulating an ACM, if it could replace the contents of any count track by that of any other in each single step. This is clearly impossible for

a one-head tape unit, since the significant count track contents may be arbitrary large. Yet we were able to update the individual tracks, with respect to unit addition/subtraction, by amortizing the propagation of the carries and borrows. The idea below is to do the same with respect to the replacement of one count track contents by that of another. Thus, if at some step t the counter i is set to the value of counter j, we start to transfer the contents of count track j to count track i, from the low order digits to the high order digits, a few digits each step t', $t' \geq t$. We do so by introducing a switch which, in each position it passes, overwrites the digit on count track i by the corresponding digit on count track j. Each such switch is introduced in the 0-th position of the representation, and is shifted through simultaneously scanned adjacent positions to higher ones, preliminary to the propagation of carries and borrows, in each step. The effect is, that the carry or borrow, resulting from an input at a later time than the input which was to replace the contents of counter i by that of counter j, will always be preceded by the replacing of the individual digits constituting the contents of count track i by their counterparts on count track j. Since in each interval of n steps, for all $n \geq 1$, the head visits but $O(\log n)$ distinct tape squares, each switch eventually overtakes all earlier created switches, but never passes them. We are thus confronted with arbitrary long queues of switches clogging at some positions on the tape. It will be shown, however, that whenever one switch overtakes another one, we can replace the combination by a single switch. It thus suffices to equip the multicounter simulator of [3] with an extra track on its tape, and modify the algorithm it executes in each step, to derive the desired ACM implementation.

2. An optimal simulation of ACM's

Recall that, in the proof of the optimal simulation of multicounter machines in [3], the usual assumption of initially zero counters was not essential. The simulation presented there also works with each counter initially set to any integer. To turn such a machine into an ACM, we therefore only have to add operations which can instantly replace the contents of any counter by that of any other counter. This amounts to an operation which is more general than a permutation of the momentary contents' amongst the various counters.

Define a *semipermutation* σ, among k objects $o_1, o_2, \ldots, o_k$, for $\sigma = i_1 i_2 \cdots i_k$

($i_j \in \{1, 2, \ldots, k\}$

for $1 \le j \le k$) by

$$\sigma(o_1, o_2, \ldots, o_k) = (o_{i_1}, o_{i_2}, \ldots, o_{i_k}) .$$

A semipermutation is also called a *permutation with repetitions.* The semigroup (not group), of which the elements are semipermutations of k objects, the product of two semipermutations being the semipermutation resulting from applying each in succession, and the identity ε being the semipermutation which does not change anything, has k^k elements and is denoted by R_k .

Define an *augmented counter machine (ACM) A* as a *k*-counter machine with each counter *i*, $1 \le i \le k$, initialized to a value in the set of integers. Input commands to *A* are of the format $(\sigma, \boldsymbol{\delta})$ with $\sigma \in R_k$ and $\boldsymbol{\delta} \in \{-1, 0, 1\}^k$. At any time, if $(c_1, c_2, \ldots, c_k)$ is the integer valued k-vector contained in *A*'s k counters, and $(\sigma, \boldsymbol{\delta})$ is the currently polled input command, then in one step *A* does all of the following:

(i) $(c_1, c_2, \ldots, c_k) \leftarrow \sigma(c_1, c_2, \ldots, c_k)$;

(ii) $(c_1, c_2, \ldots, c_k) \leftarrow (c_1, c_2, \ldots, c_k) + \boldsymbol{\delta}$;

(iii) OUTPUT, *for all i,* $1 \le i \le k$, *''counter i = 0'' or ''counter i ≠ 0'' according to the new state of affairs.*

Let *M* be the k-counter machine simulator as constructed in [3]. The i-th count track contains the array $C[i, 6{:}\infty]$, and the i-th register in the finite control contains $C[i, 0{:}5]$. The array $C[i, 0{:}\infty]$ represents the integer c_i, that is, the value of the i-th counter, $1 \le i \le k$, just as the array $C[0{:}\infty]$ represented the value c of the quintessential counter in [3]. The initial arrays $C^0[i, 0{:}\infty]$, $1 \le i \le k$, are representations of the prescribed initial integers, each representation containing no digits of opposite sign, cf. the conclusion of section 2.2 in [3]. The following adaptation, of procedure *STEP* in section 2.4 of [3], trivially turns *M* into an ACM simulator. Replace ***step* 3** of procedure *STEP* by ***step* 3_1** below, turning it into a new procedure $STEP_1$. The resulting machine is M_1, and the contents of the k count tracks are contained in a $k \times \infty$ matrix $C_1[1{:}k, 0{:}\infty]$, such that $C_1[i, 0{:}\infty]$ denotes the contents of the i-th register in M_1's finite control , followed by the i-th count track on M_1's tape, in the obvious way, $1 \le i \le k$.

***Step* 3_1**: *Let the current value of I, determined by* ***step* 2** *, be* $\{i_l, i_{l-1}, \ldots, i_1\}$ *with* $i_l > i_{l-1} > \cdots > i_1$. READ *the current command* $(\sigma, \boldsymbol{\delta})$ *from the input terminal. Let* $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_k)$. *Execute:*

for j=0 **step** 1 **until** ∞

do

$C_1[1{:}k, j] \leftarrow \sigma\, C_1[1{:}k, j]$

od ;

for j=l **step** −1 **until** 1

do for i=1 **step** 1 **until** k

do

$C_1[i,\ 2i_j : 2i_j{+}3] \leftarrow$ UPDATE $(C_1[i,\ 2i_j : 2i_j{+}3])$

od

od ;

for i=1 **step** 1 **until** k

do

$C_1[i,\ 0{:}1] \leftarrow$ INPUT $_{\delta_i}$ $(C_1[i,\ 0{:}1])$

od

***Step* 3_1**, however,contains an infinite **for** statement. (That statement is the only addition to the original ***step* 3**.) Since the cardinality of $I(t)$ happens to be at most 4, for all t, cf. [3], only a few positions of the arrays, representing the counters, can be updated by the actual machine in each step. Consequently, M_1 does not constitute an real machine, since it executes the procedure $STEP_1$, containing an infinite **for** statement, that accesses all of the infinite tape (c.q., $C_1[1{:}k,\ 0{:}\infty]$), each single step. We shall amortize the execution of the infinite **for** statements, implementing the semipermutations, by executing them in each position only when they are due.

We observe the notational conventions from [3], concerning superscripts on arrays. Thus, an array B, connected with a machine M_i, $i = 1,2$, can be viewed as a variable or as an actual value. In the first case we do not use a superscript. In the latter case a superscript t is used to

indicate the value of B, subsequent to the execution by M_i of the t-th step (i.e., procedure $STEP_i$), for a given input command sequence $(\sigma^1,\boldsymbol{\delta}^1), (\sigma^2,\boldsymbol{\delta}^2), \ldots, (\sigma^t,\boldsymbol{\delta}^t), \cdots$. That is, B's value just before M_i processes the $(t+1)$ -th input command $(\sigma^{t+1},\boldsymbol{\delta}^{t+1})$.

We associate, with each position $j\geq 0$, a *queue* $Q[j]$ of semipermutations. If $Q[j] = \sigma_m\sigma_{m-1} \cdots \sigma_1$ then the constituent semipermutations $\sigma_1, \sigma_2, \ldots, \sigma_m$ have been executed, in that order, on all positions j_1, $0\leq j_1\leq j$, but none of them has as yet been executed on any position j_2, $j_2>j$. Queues of semipermutations can be *concatenated* to a single queue. That is, if $Q[j_1] = \sigma_p\sigma_{p-1} \cdots \sigma_1$ and $Q[j_2] = \nu_q\nu_{q-1} \cdots \nu_1$ then by definition:

$$Q[j_1]\,Q[j_2] = \sigma_p\sigma_{p-1} \cdots \sigma_1\nu_q\nu_{q-1} \cdots \nu_1 .$$

For each $j\geq 0$, the *initial* contents of $Q[j]$ is ε, that is, the *empty* queue. For each particular input command sequence, for each time $t\geq 0$, we denote, for all $j\geq 0$, the queue in position j at time t by $Q^t[j]$. Thus, $Q^0[j] = \varepsilon$ for all $j\geq 0$. For any input sequence $(\sigma^1,\boldsymbol{\delta}^1), (\sigma^2,\boldsymbol{\delta}^2), \ldots, (\sigma^t,\boldsymbol{\delta}^t), \cdots$, with $\sigma^t \in R_k$ and $\boldsymbol{\delta}^t \in \{-1,0,1\}^k$, for all $t\geq 1$, we preserve the following invariant:

(E) $$\underset{t\geq 0}{t}\ [\ Q^t[0]\,Q^t[1] \cdots Q^t[j]\,Q^t[j+1] \cdots = \sigma^t\sigma^{t-1} \cdots \sigma^1\ \ \&\ \ \underset{i\geq 0}{i}\ [\ Q^t[2i] = \varepsilon\ ]\ ] .$$

(Recall that, in [3], invariants (A)-(D) pertain to the representation $C[i\ , 0{:}\infty]$ of the contents of the i-th simulated counter, for each i, $0\leq i\leq k$.)

If $Q[j] = \sigma_m\sigma_{m-1} \cdots \sigma_1$ then by *application* of $Q[j]$ to a k-vector $\boldsymbol{v} = (v_1, v_2, \ldots, v_k)$, denoted as

$$(v_1, v_2, \ldots, v_k) \leftarrow Q[j]\,(v_1, v_2, \ldots, v_k) \ ,$$

we mean the assignment embodied in the execution of:

for j=1 **step** 1 **until** m

do

$(v_1, v_2, \ldots, v_k) \leftarrow \sigma_j(v_1, v_2, \ldots, v_k)$

od

Now replace the third step of procedure *STEP* by ***step*** $\mathbf{3_2}$, so as to obtain a new procedure $STEP_2$. The corresponding machine is M_2 and, for any input sequence $(\sigma^1,\boldsymbol{\delta}^1), (\sigma^2,\boldsymbol{\delta}^2), \ldots, (\sigma^t,\boldsymbol{\delta}^t), \cdots$, with $\sigma^t \in R_k$ and $\boldsymbol{\delta}^t \in \{-1,0,1\}^k$, the matrix $C_2[1{:}k, 0{:}\infty]$ contains the contents of the k count tracks and k count registers of M_2 in the obvious way.

***Step* 3_2**: *Let the current value of I, determined in* ***step* 2**, *be* $\{i_l, i_{l-1}, \ldots, i_1\}$ *with* $i_l > i_{l-1} > \cdots > i_1$. READ *the current command* $(\sigma, \boldsymbol{\delta})$ *from the input terminal. Let* $\boldsymbol{\delta} = (\delta_1, \delta_2, \ldots, \delta_k)$. *Execute:*

for $j = l$ **step** -1 **until** 1

do

$C_2[1{:}k\,,\, 2i_j{+}2{:}2i_j{+}3] \leftarrow Q[2i_j{+}1]\, C_2[1{:}k\,,\, 2i_j{+}2{:}2i_j{+}3]$;

$Q[2i_j{+}3] \leftarrow Q[2i_j{+}1]\, Q[2i_j{+}3]$;

$Q[2i_j{+}1] \leftarrow \varepsilon$;

for $i = 1$ **step** 1 **until** k

do

$C_2[i\,,\, 2i_j{:}2i_j{+}3] \leftarrow$ UPDATE $(C_2[i\,,\, 2i_j{:}2i_j{+}3])$

od

od ;

$C_2[1{:}k\,,\, 0{:}1] \leftarrow \sigma\, C_2[1{:}k\,,\, 0{:}1]$;

$Q[1] \leftarrow \sigma\, Q[1]$;

for $i = 1$ **step** 1 **until** k

do

$C_2[i\,,\, 0{:}1] \leftarrow$ INPUT $_{\delta_i}(C_2[i\,,\, 0{:}1])$

od

Obviously, (E) is preserved by ***step* 3_2** for each input sequence.

LEMMA 1. *For each input sequence it holds that for all* $t \geq 0$ *and all* i, $1 \leq i \leq k$, *we have:*

$$C_1^t[i\,,\, 0] = \bar{0} \;\; i \quad C_2^t[i\,,\, 0] = \bar{0}.$$

Proof. Define a third $k \times \infty$ matrix C_3, which *normalizes* C_2, at any instant of time t, by

executing the backlog of semipermutations which by that time have accumulated (in the queues for) the consecutive positions j, with respect to the k-vectors $C_2^t[1{:}k\,,\,j\,]$. By definition then, for all $t \geq 0$ and all $j \geq 0$:

$$C_3^t[1{:}k\,,\,j\,] = (\,Q^t[0]\,Q^t[1]\,\cdots\,Q^t[j-1]\,)\,C_2^t[1{:}k\,,\,j\,]\,.$$

The following Claim expresses the essence of the amortization-of-execution-of-semipermutations argument. Viz., for each input sequence, for every t and i, $C_3^t\,[i,\,0{:}\infty]$ and $C_1^t\,[i,\,0{:}\infty]$ represent the same integer.

Claim. *For any particular input sequence* $(\sigma^1,\boldsymbol{\delta}^1)\,,\,(\sigma^2,\boldsymbol{\delta}^2)\,,\,\ldots\,,\,(\sigma^t,\boldsymbol{\delta}^t)\,,\,\cdots$ *:*

$$\underset{t\geq 0}{t}\ \underset{1\leq i\leq k}{i}\ \underset{c\in Z}{c}\ [\ C_3^t[i\,,\,0{:}\infty] \in \text{code}\ (c)\ \ i\ \ \ C_1^t[i\,,\,0{:}\infty] \in \text{code}\ (c)\ ]\ ,$$

where Z *is the set of integers.*

Proof of Claim. By induction on the number of steps t, for any particular input sequence $(\sigma^1,\boldsymbol{\delta}^1)\,,\,(\sigma^2,\boldsymbol{\delta}^2)\,,\,\ldots\,,\,(\sigma^t,\boldsymbol{\delta}^t)\,,\,\cdots$.

Base case: $t=0$. Since $Q^0[j\,]=\varepsilon$, for all positions $j \geq 0$, and C_1^0 and C_2^0 both represent the same k-vector of prescribed integers according to the code function, cf. [3], the Claim holds initially.

Induction: $t \geq 0$. Assume the Claim holds for all $s \leq t$. Let $I(t+1) = \{i_l, i_{l-1}\,,\,\ldots\,,\,i_1\}$, with $i_l > i_{l-1} > \cdots > i_1$. Recall from [3] that, for each $t \geq 0$, the least element i_1 in $I(t+1)$ equals 0. This will be needed later in the proof. By the inductive assumption, for each i, $1 \leq i \leq k$, and for all s, $0 \leq s \leq t$, there is an integer c_i^s such that $C_3^s[i\,,\,0{:}\infty]\,,\,C_1^s[i\,,\,0{:}\infty] \in$ code (c_i^s), since M_1 obviously simulates an ACM A just as M in [3] simulates multicounter machines. During step $t+1$, the running variable j assumes the successive values $l, l-1\,,\,\ldots\,,\,1$ in ***step*** $\mathbf{3_2}$ of procedure $STEP_2$. For each such j, the piece of code

(1) $C_2[1{:}k\,,\,2i_j{+}2{:}2i_j{+}3] \leftarrow Q[2i_j{+}1]\,C_2[1{:}k\,,\,2i_j{+}2{:}2i_j{+}3]$;

$Q[2i_j{+}3] \leftarrow Q[2i_j{+}1]\,Q[2i_j{+}3]$;

$Q[2i_j{+}1] \leftarrow \varepsilon$

in ***step* 3_2** does not change the normalized matrix $C_3[1{:}k\,,\,0{:}\infty]$ at all. The execution of (1) also preserves (E), viz., in particular $Q[i\,]=\varepsilon$, for all even i. Now consider the next piece of ***step* 3_2**:

(2) **for** $i=1$ **step** 1 **until** k

do

$C_2[i\,,\,2i_j{:}2i_j{+}3] \leftarrow$ UPDATE $(C_2[i\,,\,2i_j{:}2i_j{+}3])$

od

Just before the execution of this **for** -statement, the matrix $C_3[1{:}k\,,\,0{:}\infty]$ consisted of three sub-matrices:

$C_3[1{:}k\,,\,0{:}2i_j{-}1]$,

$C_3[1{:}k\,,\,2i_j{:}2i_j{+}3]$, *and*

$C_3[1{:}k\,,\,2i_j{+}4{:}\infty]$.

Since $Q[2i_j] = Q[2i_j{+}2] = \varepsilon$, by invariant (E), and $Q[2i_j{+}1]$ has just been set to ε by the preceding subprogram (1), it follows from the definition of C_3 that, just before execution of (2), it holds that:

(3) $C_3[1{:}k\,,\,2i_j{:}2i_j{+}3] = (Q[0]\,Q[1]\,...\,Q[2i_j{-}1])\,C_2[1{:}k\,,\,2i_j{:}2i_j{+}3]$.

Only $C_2[1{:}k\,,\,2i_j{:}2i_j{+}3]$ is accessed and changed (row-by-row) according to UPDATE in (2). Therefore, by equality (3), the effect on the normalized matrix $C_3[1{:}k\,,\,0{:}\infty]$, of executing the piece of code(2) on $C_2[1{:}k\,,\,0{:}\infty]$, is the same as the effect of executing:

for i=1 **step** 1 **until** k

do

$C_3[i\,,\,2i_j{:}2i_j{+}3] \leftarrow$ UPDATE $(C_3[i\,,\,2i_j{:}2i_j{+}3])$

od

By Propositions 1-4 in [3], therefore, if $C_3[i\,,\,0{:}\infty] \in$ code (c_i), for some integer c_i, before the execution of (2), then $C_3[i\,,\,0{:}\infty] \in$ code (c_i) after the execution of (2) too, for each i, $1 \leq i \leq k$. As noted above, for all $t \geq 0$, the least element of $I(t+1)$ is $i_1=0$, by [3]. So subsequent to the last execution of the subprogram (1);(2) in the $t+1$ -th step, that is, the execution with $j=1$ and therefore $i_j=i_1=0$, we have $Q[1]=\varepsilon$ while $Q[0]=\varepsilon$ by (E). Hence, by definition, $C_3[1{:}k\,,\,0{:}1]$ now equals $C_2[1{:}k\,,\,0{:}1]$, while, by the inductive assumption and the above reasoning, still $C_3[i\,,\,0{:}\infty] \in$ code (c_i^t), for all i, $1 \leq i \leq k$. In this situation

(4) $C_2[1{:}k\,,\,0{:}1] \leftarrow \sigma C_2[1{:}k\,,\,0{:}1]$;

$Q[1] \leftarrow \sigma Q[1]$

is executed. Thus, the array $C_3[1{:}k\,,\,0{:}\infty]$, derivable from the new values of $C_2[1{:}k\,,\,0{:}\infty]$ and $Q[0{:}\infty]$, yields, for $\sigma = j_1\, j_2\, \cdots\, j_k$, that $C_3[i\,,\,0{:}\infty] \in$ code $(c_{j_i}^t)$, for $1 \leq i \leq k$, while $Q[0]=\varepsilon$. Consequently, under the inductive assumption, after the execution of (4) in the $t+1$ -th step, we have $C_3[i\,,\,0{:}\infty] \in$ code $(c_{j_i}^t)$, just as we trivially have $C_1[i\,,\,0{:}\infty] \in$ code $(c_{j_i}^t)$, subsequent to the execution of

(5) **for** j=1 **step** 1 **until** ∞

do

$C_1[1{:}k\,,\,j\,] \leftarrow \sigma C_1[1{:}k\,,\,j\,]$

od

in the $t+1$ -th step of M_1 (using $STEP_1$ containing ***step*** $\mathbf{3_1}$). Meanwhile, we still have $C_3[1{:}k\,,\,0{:}1] = C_2[1{:}k\,,\,0{:}1]$, since $Q[0]=\varepsilon$. Therefore, subsequent to the final piece

(6) **for** i=1 **step** 1 **until** k

do

$C_2[i\,,\,0{:}1] \leftarrow \text{INPUT}_{\delta_i}(C_2[i\,,\,0{:}1])$

od

of ***step*** 3_1, yielding the new values of C_2 and C_3, viz., C_2^{t+1} and C_3^{t+1}, we still have $C_3^{t+1}[i\,,\,0{:}1] = C_2^{t+1}[i\,,\,0{:}1]$, for all i, $1 \le i \le k$. Moreover, by the properties of INPUT in [3] we also have $C_3^{t+1}[i\,,\,0{:}\infty] \in \text{code}\,(c_{j_i}^t + \delta_i)$, for all i, $1 \le i \le k$. Trivially, in view of [3], for all i ($1 \le i \le k$), it holds that $C_1^{t+1}[i\,,\,0{:}\infty] \in \text{code}\,(c_{j_i}^t + \delta_i)$. This concludes the induction and the proof of the Claim. □□

Since invariant (E) is preserved by ***step*** $\mathbf{3_2}$, and therefore $Q[0] \equiv \varepsilon$, we have by definition that $C_3[1{:}k\,,\,0{:}1] \equiv C_2[1{:}k\,,\,0{:}1]$. In [3, Proposition 2] it was shown that the lowest order digit of a representation in code (c) equals $\bar{0}$ iff c equals 0. Together with the Claim, these two observations imply the Lemma. □

Since it is trivial that M_1 real-time simulates the required ACM, by Lemma 1 it follows from [3] that, if the machine M_2 can be realized, the Theorem holds.

LEMMA 2. *M_2 can be constructed as a machine satisfying the specifications in the Theorem.*

Proof. The only difficulty with M_2 concerns the storage, execution, transport and concatenation of arbitrary large queues of semipermutations. Since the semipermutations form the semigroup R_k under concatenation, no queue $Q[j]$, $j \ge 0$, ever needs to contain more than a single element from R_k. Since every $Q[j]$, $j \ge 0$, initially contains the unity element ε, each subsequent execution of ***step*** $\mathbf{3_2}$ can compute the single semipermutation which represents the current contents of $Q[j]$ in R_k, for any such $Q[j]$ involved. Storing $Q[j]$ in the cell containing $C[1{:}k\,,\,j]$, for all $j \ge 0$, so in the finite control of M_2 for $0 \le j \le 5$ and on its tape for $j \ge 6$, shows that M_2 has the

same specifications as the multicounter simulator M in [3]. Hence the Lemma. □

The Theorem follows from Propositions 1-4 in [3] and Lemma's 1,2 above, combined with the observation that M_1 trivially real-time simulates any ACM.

3. Final remarks

Optimality. Since the ACM implementation, constructed above, has the same complexity, with respect to the measures concerned, as does the multicounter machine implementation in [3], it is *à fortiori* also optimal in all commonly considered complexity measures at once.

On the required number of bits. There are k^k semipermutations in R_k. To denote each of them, it suffices to use $k\log_2 k$ bits. Similar to [3], we note that, under the scheme outlined in section 2, it suffices to use $(4k+k\log_2 k+2)\log_2 n$ bits to represent k counts, of absolute value not greater than n, in the ACM simulator. Using a redundant symmetric r-ary representation [3], based on the digits $-r, -r+1, \ldots, 0, 1, \ldots, r-1, r$,we can bring the bit count down to to below $(1+(4+\log_2 k)/\log_2 r)k\log_2 n$ bits, and therefore arbitrary close to the information theoretical minimum, to the detriment of the implicit constant delay, as in [3].

Simulations of ACM's on other devices. In [2] we gave optimal simulations of multicounter machines on RAM's, combinational logic networks, cyclic logic networks and VLSI. The method used above, of amortizing execution of semipermutations to extend the simulation of multicounter machines by tape units to a simulation of ACM's by the same, can also be used to extend the optimal simulations of multicounter machines, by the above devices as in [2], to optimal simulations of ACM's by these devices. As here, the complexity of the simulations of the ACM's, by these devices, is none other than the complexity of the corresponding simulations of multicounter machines.

REFERENCES

1. Fischer, P.C., A.R. Meyer & A.L. Rosenberg, Counter machines and counter languages, *Math. Systems Theory* **2** (1968), 265-283.

2. Vitányi, P.M.B., Efficient simulations of multicounter machines, *Information and Control*, in press.

3. Vitányi, P.M.B., An optimal simulation of counter machines, *SIAM Journal on Computing*, this issue.