Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J. Seiferas, P.M.B. Vitányi

Counting is easy
(preliminary version)

Department of Computer Science Report CS-R8423 December

Counting is Easy
(preliminary version)

Joel Seiferas

Computer Science Department, University of Rochester
Rochester, New York 14627

Paul M.B. Vitányi

Centrum voor Wiskunde en Informatica, Amsterdam

An exposition is presented of the recent result that many independent counts with simultaneous zero-check can be maintained in real-time on an oblivious single-head tape unit using the information-theoretical storage minimum. Some extensions of that result are also given. The treatment is informal and aims at making the central ideas more transparent.

1980 Mathematics Subject Classification: 68C40, 68C25, 68C05, 94B60, 10-00
CR Categories: F.1.1, F.1.3, F.2.3, B.3.2, E.1, E.4
Keywords & Phrases: counting, number representation, coding, counter machine, multicounter machine, augmented counter machine, real-time simulation by oblivious one-tape Turing machine
Note: Parts of this report have appeared in *Dopo Le Parole,* H.W. Lenstra, Jr., J.K. Lenstra & P. van Emde Boas, Eds., Private Edition, Amsterdam, 1984.

1. Introduction

From the earliest times man has developed ever newer methods to count and denote numbers. People knew how to count long before they wrote numerals; simple arithmetic evolved slowly from the operation of counting. It may be observed that denotation for number tends to adapt to the use made thereof. The German student keeps his beer score by chalk marks on the table or nearby wall. In Roman times the familiar variant of this tally system of number representation was used. Increasing use of the arithmetic operations of addition, subtraction, multiplication and division, instead of the use of number representation as just memory aid subject to unit changes, forced the emergence of the more convenient positional notation in general use. This notwithstanding the fact that positional notation is logically considerably more complicated than the tally system. In recent times under the advent of electronic computing devices the use of all sorts of binary representations has flourished. The express purpose of such representations is to alleviate the difficulty of the task at hand. Now consider the task of counting. Tallying is admirably suited for counting, since the addition or subtraction of a unit is always performed by a unit change in the representation. Nonetheless, the already huge size of representation for quite moderate numbers makes the system unwieldy. The common positional system, on the other hand, suffers the drawback that for certain counts a change by a single unit causes more than one digit or even all digits of the representation to change. This means that we have to access unbounded information which takes unbounded time. In less common positional representations, such as Gray codes, only a single bit has to be changed to add or subtract a unit. Unfortunately, the position of this bit depends on the stored number and

Report CS-R8423
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

whether we add or subtract, forcing us to compute that position and therefore to access all of the stored number. Since in computers it is common and feasible to manipulate far larger numbers than without, the question has cropped up in the sixties whether we can combine the ease of tally update with the compactness of positional representation. An additional requirement of the looked-for representation is that the unit time for update should also suffice to check whether the number is zero or not. A device implementing these capabilities is called a *counter.* To make matters more difficult, and to get some insight in the computational complexity of several problems related to counting, the next question asks whether we can so maintain an arbitrary fixed number of such counts, that is, a *multicounter*, on a single-head tape unit [3]. Such a single head tape unit consists of a finite-state control with an input terminal and an output terminal and a single head storage tape. This device is known as a *one-tape on-line Turing machine.*
The result of an investigation [4] was that counting is *easy* in the sense of computational complexity: an arbitrary fixed number of independent counts can be maintained, under unit increments/decrements and simultaneous check for 0, by a real-time oblivious one-head tape unit in logarithmic space. *Oblivious* means that the input data do not influence the storage head movement, that is, the motion of the storage tape head is a function of time alone. The celebrated ARJEN LENSTRA has on several occasions been of the opinion that "the result cannot be true; were I an American university professor, I would tell a student to find the error". Bent on simplifying the solution, the first author in a letter [2] to the second author, in a top-down development of the ideas, demonstrated one of the possible alternatives to the original bracketing to control the inherent recursion. This incarnation of the solution takes twice as many bits (4) per tag as the original tagging scheme in [4]. It is most satisfactory however that he improved the details of the radix notation used. Below we follow this appealing exposition and omit all proofs.

2. COUNTING MADE EASY

We will maintain each counter as a logical sequence of symbols, similar to the familiar radix notation.

Theorem. *There is a data-independent (oblivious) maintenance schedule with the following properties:*

1. There is a chance to propagate information such as "unit increments" and "unit decrements" at least once in every O(1) *steps to the logical* 0*th position.*

2. There is a chance to propagate information as "carries" and "borrows" from position i to position $i+1$ at least once every O(1) *times there is a chance to propagate such information into position i, for all $i \geqslant 0$.*

3. Maintenance of this schedule can be implemented on a single-tape Turing machine with bounded delay and oblivious head motion.*

The counter contents are denoted in a *redundant symmetric* positional representation. The underlying radix notation used can be base 10, since the constants involved in the O(1) upper bounds turn out to be small (about 3) compared to 10. The representation is

* A Turing machine has *bounded delay* if it performs no more than a fixed constant number of elementary steps in between polling successive input commands from the input terminal. In a *real-time* computation this fixed constant is 1. At the cost of adding states to the finite control and expanding the storage tape alphabet, of the Turing machine concerned, a constant delay computation can always be sped up to a real-time computation.

symmetric since both positive and negative digits are used. A mark is maintained on the most significant nonzero digit (if any) and on the nonsignificant leading 0's. The representation is redundant since all nonzero integers have infinitely many representations this way, even without nonsignificant leading 0's. The rules for information propagation from position i to position $i+1$ are:

- "Carry" if the digit is greater than 5.
- "Borrow" if the digit is less than -5.
- Do nothing if the digit is bounded by 5 in absolute value.

By induction, the properties of the maintenance schedule assure that no digit will have to exceed 9 in absolute value. As a consequence, the only digit that might change from zero to nonzero, or vice versa, is at position $i+1$ above, so that only the marks at positions i and $i+1$ might have to change; thus, the marks can be correctly maintained. As another consequence, the most significant nonzero digit (if any) will always correctly indicate the sign of the entire count, so that the count will be 0 if and only if the frequently observed digit at position 0 is a marked 0.

3. The maintenance schedule

For *random* access we can use the standard Thue sequence of visits, visiting 0 every other step, 1 every other remaining step, 2 every other now remaining step, etc.

$$0102010301020104010201030102010501020103010201040102\cdots$$

Equivalently, bracket each visit to $i+1$ by review tours of of positions $0, 1, \ldots, i$. Thus,

```
tour ($i+1$):
    tour ($i$);
    visit $i+1$;
    tour ($i$)
```

and

```
tour (0):
    visit 0 .
```

We propagate the carries and borrows from i to $i+1$ on visit to i. Noting that appending 'visit $i+1$; tour (i)' onto the end of 'tour (i)' always gives 'tour ($i+1$)' we see that 'tour (∞)' can be defined as the limit of 'tour (i)' for i goes to infinity.

```
tour ($\infty$):
    visit 0;
    visit 1; tour (0);
    visit 2; tour (1);
    visit 3; tour (2);
    visit 4; tour (3);
    ...
```

4. Turing machine implementation

The storage tape of our Turing machine shall be semi-infinite to the right. The initial left-most square is called the start square. In the *instantaneous description* (momentary snapshot) of the tape of the Turing machine we mark the head position by a $>$ or a $<$. The start instantaneous description is

$$>12...i\,(i+1)...\ .$$

Since the head must always be near the logical position 0 we can try:

$$\{\ >012...i\,(i+1)...\ \}\quad \text{tour}\,(i)\quad \{\ i...210>(i+1)...\ \}\ ,$$

in preparation for a visit to $i+1$. However, now tour(i) will no longer complete the desired analogous preparation (that is, tour$(i+1)$) for a visit to $i+2$. So we construct a revision:

```
tour ($i+1$):
    tour ($i$);
    visit $i+1$;
    tour ($i$) "transporting" $i+1$;
    tour ($i$) "to get back"
```

Denoting 'tour (i) "transporting" $i+1$' as 'tour$(i)+$', we now obtain:

```
tour ($\infty$):
    visit 0;
    visit 1; tour (0)+; tour (0);
    visit 2; tour (1)+; tour (1);
    visit 3; tour (2)+; tour (2);
     . . .
```

Roughly speaking therefore:

$$\{\ ...>01...i\,(i+1)\,(i+2)...\ \}$$

tour (i) $\{\ ...i...210>(i+1)\,(i+2)...\ \}$ visit $i+1$;

tour (i) "transporting" $i+1$ $\{\ ...(i+1)<012...i\,(i+2)...\ \}$

tour (i) "to get back" $\{\ ...(i+1)\,i\,...210>(i+2)...\ \}$ visit $i+2$;

shows the effect of 'tour$(i+1)$; visit $i+2$'.

Remaining worries:

(1) How to do the "transport"?

(2) How to keep track of the recursion without a stack?

(3) When get and recognize the chance to propagate information between logical positions i and $i+1$?

To transport a symbol through a tour we have a second version of each symbol which gets pushed back whenever it gets in the way. Because the second version of $(i+1)$ will be transported, consecutively (but not contiguously) in time, across the sequence $0, 1, 2, \ldots, i$ of the original variety, a simple pushback like

$$> a^{2nd\text{ version}} b^{1st\text{ version}} \quad \Rightarrow \quad > b^{1st\text{ version}} a^{2nd\text{ version}}$$

will always suffice. The recursive formulation is as follows.

```
tour $(i+1)$ “transporting” $j^{2nd\text{ version}}$:
    tour $(i)$ “transporting” $j^{2nd\text{ version}}$;
    push $j^{2nd\text{ version}}$ back past $(i+1)^{1st\text{ version}}$;
    visit $i+1$;
    tour $(i)$ “transporting” $(i+1)^{2nd\text{ version}}$;
    tour $(i)$ “empty-handed”
```

We have to keep track of the recursion, and to know when to turn. So along with each symbol we maintain the current physical direction (left or right) to its logical successor. Assuming this is done correctly, it is clear when the last pushback of $(i+1)$ occurs, and the symbol can be so marked (the third version). This gives:

$$> d_1^{2nd\text{ version}}\ l_2^{1st\text{ version}} \quad \Rightarrow \quad > r_2^{1st\text{ version}}\ d_1^{3rd\text{ version}} \quad ,$$

suppressing all but each symbols direction indication (l for ‘left’ and r for ‘right’ or d for either). Since only the logically first symbol is not transported nontrivially after it is visited, we maintain a special mark thereon (the fourth version). Again suppressing all but the head representation and direction and each symbol’s direction indication of, respectively, “left successor” and “right successor” in the notation: (1) first version $\leftarrow$ and $\rightarrow$; (2) second version $\in$ and $\ni$; (3) third version $\Leftarrow$ and $\Rightarrow$; (4) fourth version $\hookleftarrow$ and $\hookrightarrow$; we finally arrive at the following simple rules for the single-tape Turing machine:

$> \hookleftarrow$	$\Rightarrow$	$\hookleftarrow >$	$> \hookrightarrow$	$\Rightarrow$	$\hookrightarrow >$	[visit first symbol]
$> \leftarrow$	$\Rightarrow$	$\in <$	$> \rightarrow$	$\Rightarrow$	$\ni <$	[visit and begin transport]
$> \in_1 \rightarrow_2$	$\Rightarrow$	$> \rightarrow_2 \in_1$	$> \ni_1 \rightarrow_2$	$\Rightarrow$	$> \rightarrow_2 \ni_1$	[pushback]
$> \in_1 \hookrightarrow_2$	$\Rightarrow$	$> \hookrightarrow_2 \in_1$	$> \ni_1 \hookrightarrow_2$	$\Rightarrow$	$> \hookrightarrow_2 \ni_1$	[pushback past first symbol]
$> \in_1 \leftarrow_2$	$\Rightarrow$	$> \rightarrow_2 \Leftarrow_1$	$> \ni_1 \leftarrow_2$	$\Rightarrow$	$> \rightarrow_2 \Rightarrow_1$	[last pushback]
$> \in_1 \hookleftarrow_2$	$\Rightarrow$	$> \hookrightarrow_2 \Leftarrow_1$	$> \ni_1 \hookleftarrow_2$	$\Rightarrow$	$> \hookrightarrow_2 \Rightarrow_1$	[first and last pushback]
$> \Leftarrow$	$\Rightarrow$	$< \leftarrow$	$> \Rightarrow$	$\Rightarrow$	$< \rightarrow$	[begin tour back]

and the symmetrical rules:

$\hookleftarrow <$	$\Rightarrow$	$< \hookleftarrow$	$\hookrightarrow <$	$\Rightarrow$	$< \hookrightarrow$	[visit first symbol]
$\leftarrow <$	$\Rightarrow$	$> \in$	$\rightarrow <$	$\Rightarrow$	$> \ni$	[visit and begin transport]
$\leftarrow_2 \in_1 <$	$\Rightarrow$	$\in_1 \leftarrow_2 <$	$\leftarrow_2 \ni_1 <$	$\Rightarrow$	$\ni_1 \leftarrow_2 <$	[pushback]
$\hookleftarrow_2 \in_1 <$	$\Rightarrow$	$\in_1 \hookleftarrow_2 <$	$\hookleftarrow_2 \ni_1 <$	$\Rightarrow$	$\ni_1 \hookleftarrow_2 <$	[pushback past first symbol]
$\rightarrow_2 \in_1 <$	$\Rightarrow$	$\Leftarrow_1 \leftarrow_2 <$	$\rightarrow_2 \ni_1 <$	$\Rightarrow$	$\Rightarrow_1 \leftarrow_2 <$	[last pushback]
$\hookrightarrow_2 \in_1 <$	$\Rightarrow$	$\Leftarrow_1 \hookleftarrow_2 <$	$\hookrightarrow_2 \ni_1 <$	$\Rightarrow$	$\Rightarrow_1 \hookleftarrow_2 <$	[first and last pushback]
$\Leftarrow <$	$\Rightarrow$	$\leftarrow >$	$\Rightarrow <$	$\Rightarrow$	$\rightarrow >$	[begin tour back]

At this point it may be instructive to go through part of the simulation, with the logical position numbers but noting that only the successor directions are actually available. As example we display in the Table below an initial segment of the sequence of instantaneous descriptions suppressing all but the head representation and direction and each symbol’s direction indication and version. This sequence was computer generated literally using the preceding rules.

>↪→→→→→→→→→→	→∋>∈↩→→→→→→→	→→>↩←→→→→→→→
↪>→→→→→→→→→→	→∋>↪⇐→→→→→→→	→→↩>←→→→→→→→
↪∋<→→→→→→→→→	→∋↪>⇐→→→→→→→	→→↩∈<→→→→→→→
⇒↩<→→→→→→→→→	→∋↪<←→→→→→→→	→→∈↩<→→→→→→→
⇒<↩→→→→→→→→→	→∋<↪←→→→→→→→	→→∈<↩→→→→→→→
→>↩→→→→→→→→→	⇒←<↪←→→→→→→→	→⇐←<↩→→→→→→→
→↩>→→→→→→→→→	⇒>∈↪←→→→→→→→	→⇐>∈↩→→→→→→→
→↩∋<→→→→→→→→	⇒>↪∈←→→→→→→→	→⇐>↪⇐→→→→→→→
→∋↩<→→→→→→→→	⇒↪>∈←→→→→→→→	→⇐↪>⇐→→→→→→→
→∋<↩→→→→→→→→	⇒↪>→⇐→→→→→→→	→⇐↪<←→→→→→→→
⇒←<↩→→→→→→→→	⇒↪∋<⇐→→→→→→→	→⇐<↪←→→→→→→→
⇒>∈↩→→→→→→→→	⇒⇒↩<⇐→→→→→→→	→←>↪←→→→→→→→
⇒>↪⇐→→→→→→→→	⇒⇒<↩⇐→→→→→→→	→←↪>←→→→→→→→
⇒↪>⇐→→→→→→→→	⇒→>↩⇐→→→→→→→	→←↪∈<→→→→→→→
⇒↪<←→→→→→→→→	⇒→↩>⇐→→→→→→→	→←⇐↩<→→→→→→→
⇒<↪←→→→→→→→→	⇒→↩<←→→→→→→→	→←⇐<↩→→→→→→→
→>↪←→→→→→→→→	⇒→<↩←→→→→→→→	→←←>↩→→→→→→→
→↪>←→→→→→→→→	⇒>∋↩←→→→→→→→	→←←↩>→→→→→→→
→↪∈<→→→→→→→→	⇒>↪⇒←→→→→→→→	→←←↩∋<→→→→→→
→⇐↩<→→→→→→→→	⇒↪>⇒←→→→→→→→	→←←∋↩<→→→→→→
→⇐<↩→→→→→→→→	⇒↪<→←→→→→→→→	→←←∋<↩→→→→→→
→←>↩→→→→→→→→	⇒<↪→←→→→→→→→	→←∋←<↩→→→→→→
→←↩>→→→→→→→→	→>↪→←→→→→→→→	→←∋>∈↩→→→→→→
→←↩∋<→→→→→→→	→↪>→←→→→→→→→	→←∋>↪⇐→→→→→→
→←∋↩<→→→→→→→	→↪∋<←→→→→→→→	→←∋↪>⇐→→→→→→
→←∋<↩→→→→→→→	→⇒↩<←→→→→→→→	→←∋↪<←→→→→→→
→∋←<↩→→→→→→→	→⇒<↩←→→→→→→→	→←∋<↪←→→→→→→

Table. The first 81 instantaneous descriptions.

Finally, we must indicate when the information should be propagated, such as carries or borrows. The rules above maintain the additional information that the symbols to the right or left of the head, not being transported, are in increasing logical order away from the head. So if $> \to_1 \to_2$ (or $> \to_1 \leftarrow_2$, $\leftarrow_2 \leftarrow_1 <$, $\to_2 \leftarrow_1 <$, $< \to_1 \to_2$, and so on) occurs, the symbols being confronted must be the ith and $(i+1)$st for some i, and the information can be safely propagated. Moreover, the ith and $(i+1)$st symbols do get confronted this way in every empty-handed tour$(i+1)$, at the end of the prefix empty-handed tour$(i-1)$. Since tour(∞) is a sequence of instances of tour$(i+1)$, every third one of which is empty-handed, interspersed with visits to positions higher than $i+1$ (in the Thue sequence manner), the number of chances to propagate the information into position i without such a confrontation of positions i and $i+1$ is bounded by three times the number of visits to position $i-1$ in tour$(i+1)$, or $3 \times 9 = O(1)$. (Some lookahead for chances to propagate information from position i or a more careful analysis could reduce the number of intervening chances to the promised 3; but we could also simply work in base 56 to get by with the bound 27.)

Multicounter machines. It is not difficult to see that the above Turing machine implementation can be used to maintain any fixed number k of counters. One simply divides each position into k subpositions, one for each count. Since the maintenance schedule is oblivious, the single storage tape head can do the same job to each individual count track subposition as it formerly did to the original single count track position.

Storage complexity. Analysis of the head movement learns that in *each* interval of n steps the head position covers a tape segment of size $\Theta(\log n)$, for *all* $n>1$. This irrespective of the counter contents. PETER GACS [1] has noticed that the used space can be kept to at most proportional to the logarithm of the largest counter contents. It is obvious that we have to forfeit oblivious operation then. The idea is to *skip* the visit to $i+1$ in tour$(i+1)$ if $i+1$ lies in entirely virgin territory:

```
tour $(i+1)$:
  tour $i$;
  **if** necessary **then**
    **begin**
      visit $i+1$;
      tour $(i)$ “transporting” $(i+1)^{2nd \text{ version}}$
    **end**
  tour $(i)$
```

Just one more rule is needed: $>a \Rightarrow <a$ if every counter's track on the symbol a contains a “virgin” marked 0.

Uniform Logspace. Finite Automata have the up till now unique property that any finite collection of them can be replaced by a single one. Thus the regular sets are closed under union and intersection. Since a deterministic device can be modified to accept the complement of the originally accepted set, the regular sets form a Boolean Algebra. In [6] it is noticed that this property also holds for multitape Turing machines of which the heads stay on $O(\log n)$ bounded tape segments for *each* interval of n steps, for *all* $n>1$. We called such machines *uniform* $O(\log n)$ storage bounded. As noted above, the tagging scheme has the machine access a $\Theta(\log n)$-length tape segment in each interval of n steps, for all $n>1$. It can be demonstrated that each uniform $O(\log n)$ storage bounded multitape Turing machine can be real-time simulated by this uniform $\Theta(\log n)$ storage bounded oblivious one-head tape unit. Therefore, in simultaneous real-time and UNIFORM LOGSPACE the amount of heads, tapes, the option of head-to-head jumps, the oblivious restriction, and so on, do not change the computational power.

Augmented Counter Machines. If we can, in a collection of k counters, also replace each counter contents by that of any other counter in the collection in a single step then we have an *Augmented Counter Machine* or ACM. We can formalize such a machine as a multicounter machine with the added instruction of *semipermutation* of the momentary counter contents. A semipermutation is a permutation with repetitions, that is, any total function of a finite set of elements into itself is a semipermutation. The set of semipermutations of k elements forms a semigroup (not group) and has k^k elements. In [5] it is observed that the exhibited mechanism suffices for one-tape real-time oblivious simulation of ACM's. All we need to do is enter a semipermutation in the 0th position, and transport it from a position i to $i+1$ before other information. The semipermutation is like a switch which switches the contents of the logical position it passes; two semipermutations overtaking each other are replaced by the corresponding semipermutation from the semigroup.

Semilinear Sets. A set S of n-tuples of integers is *linear* if there exist non-negative n-tuples c, $p_1, \ldots, p_r$ such that $S = \{c+\sum_{i=1}^{r} k_i p_i \mid \text{each } k_i \geqslant 0\}$. S is *semilinear* if it is a finite union of linear sets. A language L is *commutative* if all permutations of every word in L are also in L. Clearly, in recognizing a commutative language, one needs only to consider the number of occurrences of the various vocabulary symbols in a given word. Enter the number of occurrences of the ith letter of a word w over an n-letter alphabet as the ith entry of an n-vector $\#(w)$. The mapping $\# : w \rightarrow \#(w)$ is called the PARIKH-mapping of w. In [3] a result attributed to R. LAING states that a language $L = \{w \in \Sigma^* \mid \#(w) \in S\}$ over an n-letter alphabet is a finite Boolean combination of sets which are real-time recognizable by 1-CM's if, and only if, the set $S = \{\#(w) \mid w \in L\}$ is a semilinear set. By the simulation outlined above it then immediately follows that a commutative language, which maps to a semilinear set under the PARIKH-mapping, can be recognized in real-time and uniform logarithmic space by an oblivious one-head tape unit.

REFERENCES

[1] P. Gacs, as communicated in [2].

[2] J. Seiferas, Letter of July 13, 1983.

[3] P.C. Fischer, A.R. Meyer and A.L. Rosenberg, Counter machines and counter languages, *Mathematical Systems Theory* 2 (1968) 265 - 283.

[4] P.M.B. Vitányi, An optimal simulation of counter machines, *SIAM Journal on Computing,* 14 (1985).

[5] P.M.B. Vitányi, An optimal simulation of counter machines: the ACM case, *SIAM Journal on Computing,* 14 (1985).

[6] P.M.B. Vitányi, On the simulation of many storage heads by one, Special issue on ICALP '83, *Theoretical Computer Science,* to appear.