Impossibility of 'Essential' Real-Time Garbage Collection in the General Case

D. Beauquier[1] **T. Crolard**[1] **A. Durand**[1] **A. Slissenko**[1♯]

Laboratory for Algorithmics, Complexity and Logic (LACL)
University Paris 12

{beauquier,crolard,durand,slissenko}@univ-paris12.fr

Abstract

We give an example of a computational problem for which no RAM (random access machine) can ensure a non-trivial garbage collection though a large amount of used memory can be in principle freed starting from some time moment. This example is of theoretical nature and means only that there can be no universal algorithm for 'sufficiently good' real-time garbage collection applicable in all situations.

1 Introduction

We give an example of computational problem solvable in real-time and such that a simple real-time RAM that solves it starts from a certain time moment to 'liberate in principle' large pieces of memory. That means that it will not use these pieces of memory for its future work. However, non-trivial garbage collection is impossible for this problem whatever be any other RAM solving it. To describe the latter property we introduce an adequate space measure for real-time RAMs. This result shows only principle difficulties to construct a universal, always applicable algorithm for real-time garbage collection and underlines the necessity to exploit particular properties of concrete problems to develop practical real-time garbage collectors.

Garbage collection is an automatic reclamation of computer storage [1]. The reader can find a good survey of this subject in [5, 6], and address to [2] for further reading on various advanced topics in garbage collection. Our result draws attention to the importance of theoretical analysis of domains of applicability of real-time garbage collectors.

In the next section 2 we give the notions we exploit, in particular, the space measure for real-time algorithms to model garbage collection and the problem of computing a value of a Boolean function (CVBF). Then we illustrate an 'ideal garbage collector' in terms of Kolmogorov–Schönhage algorithms[2] to get some estimation of the needed memory. Our result and a proof are in section 3.

[1]*Address:* Dept. of Informatics, University Paris 12, 61 Av. du Gén. de Gaulle, 94010, Créteil, France

♯ Member of St-Petersburg Institute for Informatics and Automation, Russian Academy of Sciences, St-Petersburg, Russia.

[2]A. N. Kolmogorov introduced the first version of his pointer machines using topological terms in 1953 [3]; the storage was a graph of bounded degree. In his lectures in the 60ies he used a notion (formally more general) where the storage was a digraph with unbounded fan-in. The latter notion was independently introduced by A. Schönhage in 1970 [4] under the name of *storage modification machine (SSM)*. Later A. Schönhage proved several remarkable theorems related to this notion (in particular, linear time multiplication by SSM, real-time simulation of multi-dimensional Turing machines by SSMs).

2 Space Measure and Computational Problem

As a model of computation we consider RAM (Random Access Machine) – any version poly-time equivalent to Turing machines will suit. We assume that the registers of a RAM can contain strings over alphabet $\mathbb{B} =_{df} \{0,1\}$ that are interpreted as integers. We permit multiplication/division and Boolean operations under the constraint that the length of registers is bounded by $\log n + O(1)$, where n is the length of the input, – this gives rather powerful model for real-time computations. By *real-time computation* we mean a computation that makes no more than a constant number of steps to process each current 'component' of the input (that can be viewed as written on an input tape that is one directional and read-only). Usually a component is a string of a fixed length, for example, a character – this will be our components. If we admit as input alphabet the alphabet $\mathbb{B} = \{0,1\}$ then we take as components strings of some fixed length (length 2 will be sufficient).

Remind that space measure for RAMs is defined as the maximal register used during computation. For example, if we used registers 1, 17 and 100, then the space size is 100. It is not the number of registers (in our example – 3) because the number of a register contains also an information that can be exploited to accelerate the computation. In other words, the used memory is defined always as the minimal initial segment of natural numbers that contains all the used registers.

Space measure for real-time garbage collection: GCMem**.**
Garbage collection concerns some kind of freeing the memory that will not be used in further computations. Take a time moment t. We will refer to time moment strictly smaller that t as *before* t, and to time moments that are greater or equal to t as *after* t.

What memory is needed for an algorithm $\mathcal{A}$ after a time moment t? Better to formulate this question as "What memory cannot be used by other processes at time t?" The memory to discuss is constituted by registers into which $\mathcal{A}$ wrote before t and by registers from which $\mathcal{A}$ reads after t. But if after t there was a writing into a register before reading there is no need to count it at t – its contents will be defined at the further moment of writing even if we free it at t. So we count only registers from where $\mathcal{A}$ read after t without writing into them before reading. Now imagine that $\mathcal{A}$ wrote before t into some 100 registers but read only from 3 of them. Clearly, only 3 registers are necessary for the functioning of $\mathcal{A}$, others can be used by other processes. But if $\mathcal{A}$ wrote into 3 registers before t and reads from 100 registers after t we cannot use these 100 registers for other processes. So only the latter registers are important to define the necessary memory.

Definition. Thus, we define $\mathrm{GCMem}_{\mathcal{A}}(W,t)$ for an algorithm $\mathcal{A}$ that processes an input W as the maximal register that was used by $\mathcal{A}$ for reading from it after t without writing into it between t and the first moment of reading after t.

Problem: Calculating a Value of a Boolean Function (CVBF).
The problem we consider has the following inputs. Each input consists of two parts, namely, of a Boolean function and of argument. The first part of the input will be called *function* and the second part will be called *query*. We will take a straightforward table representation of Boolean functions as it is not so essential. So the function part of an input is a list of 2^n (for an arbitrary n) distinct strings of the form $(a_1 \dots a_n, b)$, where $a_i, b \in \mathbb{B}$. Here list $(a_1 \dots a_n)$ represents an argument of a Boolean function and b represents its value for this argument. The query part of the input is an argument of the Boolean function, i. e. a list $(a_1 \dots a_n)$, where $a_i \in \mathbb{B}$. The *problem* is to output the value of the function for the argument given by the query.

And our goal is to give a lower bound to the space measure introduced above. To compare our lower bound with the memory 'really needed' to solve the problem of CVBF we give its *solution* *by a Kolmogorov–Schönhage algorithm*, that we will denote $\mathcal{A}_{KS}$. Algorithm $\mathcal{A}_{KS}$ when reading the function part of an input constructs a usual complete binary tree whose leaves are labeled by the values of the function. A query is processed in a straightforward way: while reading of the query $\mathcal{A}_{KS}$ cuts off all redundant branches of the tree, so after having read l bits of the query it keeps only a subtree of the depth $(n-l)$ whose size (of representation in the memory) is $O(2^{(n-l)})$. The latter value is the memory that cannot be freed by this moment of time.

Denote by KSspace the space measure for Kolmogorov–Schönhage algorithms: $\mathrm{KSspace}_{\mathcal{A}}(W,t)$ is the number of vertices of the memory graph of algorithm $\mathcal{A}$ reachable from the control (active) vertex at time t when processing an input W.

3 Impossibility Result and Its Proof

We can formulate our result in terms of the two space measures introduced above.

Main Theorem. *There are real-time computational problems such that for any RAM $\mathcal{A}$ solving such a problem and for some Kolmogorov–Schönhage algorithm $\mathcal{B}$ solving the same problem there are inputs W and time moments t such that*

$$\frac{\mathrm{KSspace}_{\mathcal{B}}(W,t)}{\mathrm{GCMem}_{\mathcal{A}}(W,t)} \leq O\left(\frac{\log^2 N}{N}\right),$$

where N denotes the length of the input W.

Proof.
Let α be any RAM that solves the problem of CVBF and consider its work on the set $2^{\mathbb{B}^n}$ of Boolean functions of n arguments.

Denote by L the maximum of $\mathrm{GCMem}_{\alpha}(W,t)$ over all W for time t corresponding to the end of processing of function part of input.

During the steps of processing of the first k bits of the query part of input algorithm α can change $O(k)$ registers (here we use the fact that α is a real-time algorithm). This change can be described as a string of the form $((r_1,v_1),\dots,(r_s,v_s))$, where $s=O(k)$, r_i is a register number and v_i is the last new value put into r_i by the end of processing of the kth bit of query. For 2^k possible values of k first bits of a query we can code possible changes of this form as a binary tree of height k whose each leaf is supplied with a list $((r_1,v_1),\dots,(r_s,v_s))$.

A Boolean function of n arguments can be coded with the help of the following informations: (i) numbers n and k, $1 \leq k \leq n$, (ii) the list $(w_0,\dots,w_L)$ of the contents of registers $0,\dots,L)$ of α that are produced when α process queries concerning this function, (iii) a tree T of height k whose leaves are supplied with lists $((r_1,v_1),\dots,(r_s,v_s))$ as described above, (iv) a value of the instruction counter of α.

Denote by β the following RAM that calculates values of a Boolean function from the just described code as follows. Machine β receives as input a code of a Boolean function of the form described above and an argument of this function. It puts in its first L registers the contents $(w_0,\dots,w_L)$, memorizes the tree T and starts to process the query. This processing follows the branch of T corresponding to the k-prefix of the argument and thus arrives at a list of the form $((r_1,v_1),\dots,(r_s,v_s))$. Then β writes v_i into r_i. After that β launches (more precisely, simulates) α starting with the instruction given by the instruction counter. Here we presume that the program of α is 'included' into the program of β. Remark that β is not demanded to be real-time.

The size of such a coding of Boolean functions is bounded from above by

$$(L + O(2^k \cdot k) + O(\log n) + O(1)). \tag{1}$$

Each of two machines α and β computes the values of Boolean functions of n arguments for respective codings. We wish to estimate from below the length of coding of Boolean functions of n arguments with respect to β.

The standard counting argument implies that for sufficiently large n at least one function, say, f has the code of the size greater than $2^n - \Delta$, where the constant Δ depends only on β (in fact, Δ is the bit length of β, and the lower bound takes place for almost all functions). This lower bound and (1) give for f the bound

$$(L + O(2^k \cdot k) + O(\log n) + O(1)) \geq (2^n - \Delta)$$

that can be rewritten as

$$L \geq 2^n - C(\cdot 2^k \cdot k - \log n - 1), \tag{2}$$

for some constant C. Take $k = n - \log n$. Then the memory of KS-algorithm solving the problem of CVBF after processing k bits of query is $O(n)$ (as the depth of the remaining tree is only $\log n$, and the GCMem of α remains exponential in n – see (2). So the rate of memories of the KS-algorithm and α is bounded from above by

$$O\left(\frac{n}{2^n - C(\cdot 2^k \cdot k - \log n - 1)}\right) \leq O\left(\frac{n}{2^n} \cdot \frac{1}{1 - \frac{C}{2^n}(\cdot 2^k \cdot k - \log n - 1)}\right) =$$

$$\frac{n}{2^n} \cdot O\left(1 + \frac{C}{2^n}(\cdot 2^k \cdot k - \log n - 1) + \frac{C^2}{2^{2n}}(\cdot 2^k \cdot k - \log n - 1)^2 + \dots\right) \leq O\left(\frac{n}{2^n}\right). \tag{3}$$

(To get this bound it suffices to choose k that insures $\frac{C}{2^n}(\cdot 2^k \cdot k - \log n - 1) \leq \theta$ with $0 < \theta < 1$.)

Remark now that the length of the input is $N = O(n2^n)$ (because of our table representation of Boolean functions). So in terms of the input length $n = O(\log N)$ and $2^n = \frac{N}{n} \geq c \cdot \frac{N}{\log N}$. From here and (3) we get the estimation of the Main Theorem. ■

References

[1] Andrew W. Appel. Garbage collection. In Peter Lee, editor, *Topics in Advanced Language Implementation*, pages 89–100. MIT Press, 1991.

[2] Yves Bekkers and Jacques Cohen, editors. *Proceedings of International Workshop on Memory Management*, volume 637 of *Lecture Notes in Computer Science*, St Malo, France, 16–18 September 1992. Springer-Verlag.

[3] Kolmogorov. O ponyatii algoritma (on the concept of algorithm). *Uspekhi Matem. Nauk*, 8(4(56)):175–176, 1953. In Russian.

[4] A. Schönhage. Universelle Turing Speicherung. In *Automatentheorie und Formale Sprachen*, pages 369–383. Bibliogr.Instutut, Mannheim, 1970.

[5] Paul R. Wilson. Uniprocessor garbage collection techniques. In Bekkers and Cohen [2].

[6] Paul R. Wilson. Uniprocessor garbage collection techniques. Technical report, University of Texas, January 1994. Expanded version of the IWMM92 paper.