Information Processing Letters 115 (2015) 193–198

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Arbitrary sequence RAMs

Michael Brand

Faculty of IT, Monash University, Clayton, VIC 3800, Australia

ARTICLE INFO

Article history:
Received 17 October 2013
Received in revised form 14 September 2014
Accepted 14 September 2014
Available online 21 September 2014
Communicated by A. Muscholl

Keywords:
Arbitrary number
Random Access Machine
Arithmetic complexity
Computational complexity
Theory of computation

ABSTRACT

It is known that in some cases a Random Access Machine (RAM) benefits from having an additional input that is an arbitrary number, satisfying only the criterion of being sufficiently large. This is known as the ARAM model. We introduce a new type of RAM, which we refer to as the Arbitrary Sequence RAM (ASRAM), that generalises the ARAM by allowing the generation of additional arbitrary large numbers at will during execution time. We characterise the power contribution of this ability under several RAM variants. In particular, we demonstrate that an arithmetic ASRAM is more powerful than an arithmetic ARAM, that a sufficiently equipped ASRAM can recognise any language in the arithmetic hierarchy in constant time (and more, if it is given more time), and that, on the other hand, in some cases the ASRAM is no more powerful than its underlying RAM.

1. Introduction

The Random Access Machine, or RAM (see [1] for a formal definition) is a computational model that affords all that we expect from a modern computer in terms of flow control (loops, conditional jump instructions, etc.) and access to variables (direct and indirect addressing). It is denoted by RAM[*op*], where *op* is the set of operations that are assumed to be executable by the RAM in a single unit of time each. A comparator for equality is also assumed to be available, and this also executes in a single unit of time. The variables (or *registers*) of an integer RAM contain elements of $\mathbb{Z}^*$, the nonnegative integers, and are also indexable by addresses that are nonnegative integers.

To discuss the power of RAMs, let us consider RAMs as calculating functions. We initialise the RAM by storing the input value, *inp*, in the RAM's $R[0]$ register (setting all other registers to zero), and the output of the function is taken to be the value of $R[0]$ at termination time. This definition can be extended to functions receiving any fixed number of inputs. Alternatively, RAMs can be discussed as language acceptors, where *inp* is taken to be in the language if and only if the return value is non-zero. Traditionally, when viewing the RAM as an acceptor, non-termination is taken to mean rejection of the input. By contrast, when viewing the RAM as a function calculator, non-termination is usually taken to mean that the RAM calculates a partial function, rather than a function. For simplicity of presentation we use the term "function" here also when referring to partial functions.

Ben-Amram and Galil [2] write "The RAM is intended to model what we are used to in conventional programming, idealized in order to be better accessible for theoretical study." However, in practice, the RAM's ability to manipulate very large numbers in constant time has been shown to reduce algorithmic complexities beyond what is usually considered "reasonable". For example, it was shown regarding many RAMs working with fairly limited instruction sets that they are able to recognise any PSPACE problem in deterministic polynomial time [17,7,3,15], and a unit-cost RAM equipped only with arithmetic operations, Boolean operations and bit shifts can, in fact, recognise in constant time any language that is recognised by a TM in time and/or space constrained by *any* elementary function of

E-mail address: michael.brand@alumni.weizmann.ac.il.

http://dx.doi.org/10.1016/j.ipl.2014.09.010
0020-0190/

the input size [4], where, for the TM, 'input size' refers to the bit length of the input integer.

In most cases (e.g., [12,10,13,11,18,14]), the large integers to be efficiently manipulated by the RAM are generated to precise values that are conducive to the computation at hand. However, in other cases (e.g., [5,9,3]) some of the integers manipulated are arbitrary, subject only to the condition of being sufficiently large. We refer to such arbitrary large numbers as ALNs.

Recently, in [4], a general framework was proposed for the study of the power contribution of ALNs, this being the ARAM. An ARAM[*op*] program, r_A, calculating a function from $\mathbb{Z}^*$ to $\mathbb{Z}^*$, is defined by an underlying RAM[*op*] program, r, that calculates a function from $\mathbb{Z}^{*2}$ to $\mathbb{Z}^*$, in the following way.

We say that r_A outputs $t \in \mathbb{Z}^*$ on input $inp \in \mathbb{Z}^*$ if there exists an $N \in \mathbb{Z}^*$ such that for any $A > N$ the output of r on (inp, A) is t, independent of A, and such that the execution time of r on input (inp, x) is bounded over all choices of x (including x values not larger than the chosen N). When this is the case, we say that the execution time of r_A on inp is the execution time of r on (inp, x) with the worst-case choice of x. In all other cases, r_A is said not to terminate.

The RAM r's second input parameter is said to be the ARAM's ALN parameter. The underlying RAM is able to compute the same function as the ARAM, and at least as quickly, if it is given as its second input a value, A, that is in a set $\{x : x > N\}$, for an appropriately-chosen N. Let us therefore define any set of the form $\{x : x > N\}$ for any N to be an "ALN set".

We now introduce a new computational model which generalises the ARAM, this being the Arbitrary Sequence RAM (ASRAM). To define the ASRAM, let us first generalise and formalise the notion of the ALN set as it pertains to ASRAMs. For this purpose, we define the Arbitrary Large Sequence (ALS) set.

Definition 1 *(ALS set).* An *Arbitrary Large Sequence (ALS) set* is a nonempty set of (infinite) integer sequences, **S**, such that for any i and any sequence $\{A_k\} \in \mathbf{S}$ there exists a sequence $\{B_k\} \in \mathbf{S}$ such that $B_i = A_i + 1$, and if $j < i$, then $B_j = A_j$.

The definition of the ALS set is such that if any finite list of integers appears as a prefix of any sequence in **S**, the last integer can be increased by 1 (and, by induction, can be replaced by any larger number), and the result would still be a prefix of a sequence in **S**. This being the case, any finite list of integers appearing as a prefix of any sequence in **S** remains a prefix in **S** if one extends it by another element, given that this element exceeds some threshold value. Other than being "large enough", the new element can be chosen arbitrarily.

Definition 2 *(ASRAM).* An *ASRAM* is a computational model that provides the same functionality as the RAM, but also allows calls to a function, "*ALN*()", that returns integers.

An ASRAM is said to output $t \in \mathbb{Z}^*$ on input $inp \in \mathbb{Z}^*$ if there exists an ALS set, $\mathbf{S} = \mathbf{S}(inp)$, such that for all $\{A_i\} \in \mathbf{S}$, if the i'th invocation of *ALN*() is replaced by the constant A_i then the resulting RAM outputs t on input inp.[1]

The run time of the ASRAM on a given inp is the run time of the underlying RAM with the worst-case choice of $\{A_i\} \in \mathbb{Z}^{*\omega}$ (regardless of whether this A_i is in **S** for any ALS set **S**). If this worst-case is unbounded, the ASRAM is taken to be non-terminating and does not produce any output, even if the condition of the previous paragraph holds.

This definition reflects a situation where every application of *ALN*() returns a number that is arbitrary other than being sufficiently large with respect to everything that occurred in earlier steps of the ASRAM's execution.

The ASRAM can be used to investigate a scenario in which an unbounded number of ALNs are required. However, we can also use it for the intermediate scenario, where only a predefined number (e.g. 2) of ALNs are available to the algorithm. This is simply done by limiting the number of times the *ALN*() function can be invoked. The original ARAM is an ASRAM limited to use *ALN*() at most once.

2. Arithmetic complexity

At face value, one may believe that multiple arbitrary numbers are no more powerful than a single arbitrary number. However, this is not so.

In this section we show that the extra power of arbitrary sequences is present already in the traditional arithmetic complexity model, this being the computational model in which the basic operations used are the four arithmetic functions, $\{+, \dot{-}, \times, \div\}$, where $a \dot{-} b \stackrel{\text{def}}{=} \max(a - b, 0)$ and "$\div$" is integer division. We also use "mod" freely in the arithmetic model, because it is an operation straightforward to simulate using the available operations.

We stress that despite use of the name "arithmetic", the results presented rely heavily on the non-arithmetic nature of "$\div$". In the literature [9], this operation is, in fact, sometimes referred to as non-arithmetic division.

Formally stated, what we prove is the following theorem.

Theorem 1. *The class of functions that can be computed in polynomial time by an arithmetic ASRAM is strictly larger than the class of functions that can be computed in polynomial time by an arithmetic ARAM.*

Note that complexity, for RAMs, is measured against the bit-length of their input integers.

To prove Theorem 1, consider first the following lemma.

[1] "*ALN*()" is a "function" only in the formal sense that it can appear in the code of an ASRAM in places where, for a RAM, one would expect a function receiving zero parameters and returning an integer. *ALN* is not a function that can actually be computed. Furthermore, it is multi-valued, in the sense that each time it is invoked it returns a new A_i, despite the fact that it is always called with the same (empty) set of parameters.