Technical Report No. 99-429

On the Power of Real–Time Turing Machines: k Tapes Are More Powerful than $k-1$ Tapes*

Stefan D. Bruda and Selim G. Akl
Department of Computing and Information Science
Queen's University
Kingston, Ontario, K7L 3N6 Canada
Email: {bruda,akl}@cs.queensu.ca

July 9, 1999

Abstract

We show that, for any integer k, there is at least one language which is accepted by a k-tape real–time Turing machine, but cannot be accepted by a $(k-1)$-tape real–time Turing machine. We show therefore that the languages accepted by real–time Turing machines form an infinite hierarchy with respect to the number of tapes used.

1 Introduction

The *real–time Turing machines* were introduced in [19]. They, and the languages accepted by them, called *real–time definable languages*, were further studied in many papers, e.g., [11, 15, 17, 18]. The model used is a deterministic one, but nondeterministic extensions were also studied, like the real–time Turing machines with restricted nondeterminism [12], and nondeterministic real–time Turing machines [7] (the languages accepted by the latter model being called *quasi–real–time languages*).

Despite the many researchers that studied these machines, one problem is, to our knowledge, still open, namely whether an addition to the number of tapes of a real–time Turing machine increases its computational power. A partial answer was given in [15], where it is shown that a 2-tape real–time Turing machine is strictly more powerful than a 1-tape one. We address in this paper the general

*This research was supported by the Natural Sciences and Engineering Research Council of Canada.

problem. We show that, indeed, for any integer k, a k-tape real–time Turing machine is strictly more powerful than a $(k-1)$-tape one.

We present in the next section a concise introduction to real–time Turing machines and real–time definable languages. Then, in section 3, we show that real–time Turing machines form an infinite hierarchy with respect to the number of tapes used. We conclude in section 4.

2 Real–Time Turing Machines

Given some alphabet A, the set A^k is defined recursively by

$$\begin{aligned} A^1 &= A \\ A^i &= A \times A^{i-1}, i > 1. \end{aligned}$$

We assume that the reader is familiar with the concept of Turing machine, therefore we do not define the terms that are usually covered in a textbook on such a subject (e.g., [13]). A Turing machine M is said to accept some language L if, for any input string w, M stops (that is, M reaches the halt state h) iff $w \in L$ [13]. The empty word is denoted by λ.

We will use the definition of real–time Turing machines presented in [17]:

Definition 2.1 [17]

1. For some constant k, $k \geq 1$, an *on–line Turing machine* is a deterministic $(k+1)$-tape Turing machine (with k working tapes and one input tape) $M = (K_p, K_a, \Sigma, W, \delta, s_0)$, where $K_p \cup K_a$ is the set of states, not containing the halt state h, s_0 is the initial state, Σ is the input alphabet, W is the alphabet of working symbols, containing the blank symbol $\#$, and δ is the state transition function, $\delta : (K_p \times \Sigma \times W^k) \cup (K_a \times W^k) \longrightarrow (K_p \cup K_a \cup \{h\}) \times (\{R, L, N\}^k \times W^k)$. The head on the input tape is allowed to move only on the right.

   A *configuration* of an on–line k-tape Turing machine is an $(k+2)$-tuple $C = (q, t, x_1\underline{a_1}y_1, \ldots, x_k\underline{a_k}y_k)$, where q is a state, $t \in \Sigma^*$ is the (not yet considered) content of the input tape, for any i, $1 \leq i \leq k$, $x_ia_iy_i$ is the content of the i-th working tape, and a_i is the symbol that is currently scanned by the head of tape i. If a configuration C_1 yields to other configuration C_2, we write $C_1 \vdash_M C_2$. As usual, $\vdash_M^*$ denotes the transitive and reflexive closure of $\vdash_M$.

   The set of states is divided into two subsets: the set of *polling* states K_p and the set of *autonomous* states K_a. All the states that lead to h in one step are polling states, and the initial state is a polling state. In addition, the relation $\vdash_M$ has the following property: if $q \in K_p$, $q'' \in K_a$, and $q' \in K_p \cup K_a$, then

$$
\begin{aligned}
(q, abv, x_1, \ldots, x_k) &\vdash_M (q', bv, x'_1, \ldots, x'_k), \\
(q'', abv, x_1, \ldots, x_k) &\vdash_M (q', abv, x'_1, \ldots, x'_k), \\
(q, \lambda, x_1, \ldots, x_k) &\vdash_M (h, \lambda, x'_1, \ldots, x'_k).
\end{aligned}
$$

M accepts the input w iff $(s_0, w, x_1, \ldots, x_k) \vdash_M^\tau (h, \lambda, x_1, \ldots, x_k)$, where $\tau > 0$ is called the *running time* of M on w.

2. A *real–time Turing machine* is an on–line Turing machine for which $K_a = \emptyset$. A language accepted by such a machine is called a *real–time definable language.*

□

In other words, an on–line Turing machine has a unidirectional input tape. Therefore, it has no knowledge about further input data. Between reading two input symbols, such a machine is allowed to go into a number of autonomous states, where it performs some work without considering any input. In addition to these requirements, a real–time Turing machine has no autonomous state, it being forced to consume an input datum at every step.

Definition 2.2 A nondeterministic real–time Turing machine is a machine that is identical to the one defined in definition 2.1, except that $\delta \subseteq ((K_p \times \Sigma \times W^k) \cup (K_a \times W^k)) \times ((K_p \cup K_a \cup \{h\}) \times (\{R, L, N\}^k \times W^k))$. The languages accepted by nondeterministic real–time Turing machines are called *quasi–real–time* languages [7]. □

3 k Tapes Are More Powerful than $k-1$ Tapes

In the following, given a word u, u^r denotes the reversal of u. For a fixed k and given some alphabet Σ, and two symbols \$ and @, $\$, @ \notin \Sigma$, let us consider the language

$$
\begin{aligned}
L = \; & \{\$w_1\$w_2\$w_3\$...\$w_k@u^r | w_1, \ldots, w_k, u \in \Sigma^*, \\
& \text{there is some } i, 1 \le i \le k, \text{ such that } u = w_i\}.
\end{aligned} \tag{1}
$$

Given a word w in L, we assume that it has the form $w = \$w_1\$w_2\$w_3\$...\$w_k@u^r$. We also denote by w_{ij} the j-th symbol of the sub-word w_i, $1 \le i \le k$, and by u_j^r and u_j the j-th symbol of u^r and u, respectively. The length of some word x is denoted by $|x|$.

Lemma 3.1 *There is a k-tape (not counting the input tape) deterministic real–time Turing machine, with the working alphabet $\Sigma \cup \{\$, @\}$ that accepts L.*

Proof. Such a machine M works as follows. While reading the input w, it writes each subword $\$w_i$, $1 \leq i \leq k$, on tape i. That is, it starts by writing the initial \$ symbol on the first tape. Then, when the head on the input tape reads some symbol w_{ij}, M writes w_{ij} on its i-th tape, and advances that tape's head on the right. When the \$ symbol that terminates the subword w_i is read, M moves the head of tape i one cell left, and, at the same time, writes \$ on the tape $i+1$. Clearly, writing the word w_i requires precisely $|w_i|$ time.

After the @ symbol is read, M starts comparing simultaneously the currently read symbol u_j with the subwords stored on each of its k tapes. For any tape i, if $u_j = w_{ij}$, then the machine advances the head of tape i one cell left; otherwise, it writes @ on tape i, and never moves the head of that tape. This step also takes $|u|$ time.

At the end of the input, on a tape i for which $u = w_i$, it is clear that the head advances each time an input symbol is read, and when the input is exhausted, the head points to the initiating \$ sign. On the other hand, assume that $|u| > |w_i|$. Then, after exhausting the word w_i, M eventually compares \$ with some symbol in Σ, which are obviously different. Then, it writes @ on tape i and never moves that head. Analogously, if $|u| > |w_i|$, then the head points to some symbol from w_i when the end of the input is reached, which is not \$. Finally, on those tapes i where there is a j such that $u_j \neq w_{ij}$, M writes @ and never moves the head. Therefore, at the end of the input, M accepts the input iff the head of at least one tape points to a \$ sign. As shown above, this happens iff there is some subword w_i such that $u = w_i$.

Moreover, it is clear that M reads one input symbol in each state, therefore M is a real–time Turing machine. $\square$

Lemma 3.2 *There is no $(k-1)$-tape (not counting the input tape) deterministic real–time Turing machine that accepts L, even if the working alphabet of M is $(\Sigma \cup \{\$, @\})^{k'}$ for some k' that depends on k.*

Proof. Assume that there exists such a machine, and denote it by M. M has $k-1$ tapes but there are k subwords w_i. However, all the subwords have to be stored somewhere, since u can match either of them. Therefore, there is some tape i that contains two subwords w_l and w_m, $l < m$. Let $o = |w_l|$ and $p = |w_m|$.

Assume now that these words are stored such that the symbols w_{lo} and w_{mp} are not stored in the same cell on tape i, and, when u_1^r is read, the head on tape i points to w_{lo}. But then, u cannot be compared with w_m in real–time, since M has to spend some time moving the head to the cell containing w_{mp} without consuming any input. Clearly, this is not possible if all the states of M are polling states.

That is, the only configuration of the tape i immediately before u^r is read, and conforming to which M is still able to compare the word u with both w_l and w_m in real–time, has the following form, where each tape cell stores a tuple of two symbols:

	w_{l1}	w_{l2}		$w_{l(o-1)}$	w_{lo}	
$\cdots$	w_{mj}	$w_{m(j+1)}$	$\cdots$	$w_{m(p-1)}$	w_{mp}	$\cdots$

Here, we showed the case $p > o$, but the format is similar for the other cases. In such a configuration, by slightly altering the machine built in the proof of lemma 3.1, we can construct a machine M that can readily compare u with all the subwords w_i, $1 \leq i \leq k$.

However, this case is impossible to achieve in real–time. Indeed, for reaching such configuration on tape i, after writing w_l and before beginning writing w_m, M has to move the head on that tape p cells left. But for any value of l and m, one can find an input word such that $\sum_{l<j<m} |w_j| < p$. But in this case M simply has no time to move p times the head on tape i.

Therefore, there are input words in L that cannot be accepted in real–time, and hence we completed the proof. □

Lemma 3.3 *There is a one-tape (not counting the input tape) nondeterministic real–time Turing machine that accepts L.*

Proof. The machine nondeterministically guesses that subword w_j that matches u, writes it on its working tape, and then compares u with the stored subword. □

The main result of our paper follows immediately from lemmas 3.1, 3.2, and 3.3:

Theorem 3.4 *For any integer k, there is at least one (quasi–real–time) language which is accepted by a k-tape real–time Turing machine, but cannot be accepted by a $(k-1)$-tape real–time Turing machine. The languages accepted by k-tape real–time Turing machines form therefore an infinite hierarchy with respect to k.* □

4 Conclusions

We showed in this paper that, for any integer k, a k-tape real–time Turing machine is strictly more powerful than a $(k-1)$-tape one. We actually began thinking about this problem as a result of a challenge that was offered to one of the authors [16]:

> Can one find any problem that is solvable by an algorithm that uses k processors, $k > 1$, and is not solvable by a sequential algorithm, even if this sequential algorithm runs on a machine whose processor is k times faster than each of the k processors used by the parallel implementation?

There are some noteworthy results in the area of parallel algorithms for real–time computations [2, 3, 4, 5, 6, 8, 9, 10, 14], but none appears to address the above challenge. A positive answer to the question for $k = 2$ is provided by (a slightly modified version of) the *pursuit and evasion on a ring* example presented in [1]. In this version, an entity A is in pursuit of another entity B on the circumference of a circle, such that A and B move at the same speed; clearly, A *never* catches B. Now, if two entities C and D are in pursuit of entity B on the circumference of a circle, such that each of C and D moves at $1/x$ the speed of A (and B), $x > 1$, then C and D *always* catch B. A computational analog of this example consists of two streams of input, both of them having to be monitored in order to solve a problem. Thanks to parallelism, two (slow) processors C and D operating simultaneously invariably succeed in completing the computation, whereas a single processor A which is twice as fast as either C or D always fails. This example can be easily extended to cases where $k > 2$.

Of course, a lot depends on one's definition of the word "solvable". Thus, if one replaces "algorithm" by "Turing machine" and "processor" by "tape" (since this is the only thing in a Turing machine that can grow as desired), then theorem 3.4 also represents a positive answer to the above challenge. Indeed, even if we restrict the working alphabet of a k-tape real–time Turing machine to the input alphabet Σ, but allow the one-tape Turing machine to use $\Sigma^{k'}$ as working alphabet, $k' \geq k$, hence giving to the unique working tape a k'-fold improvement in performance, the one-tape Turing machine is still not able to handle the acceptance of the language described in equation (1).

Granted, it is questionable whether a k-tape Turing machine is equivalent in some sense to a k-processor machine. Therefore, we conclude this paper with a new challenge: Either prove or disprove the mentioned equivalence (between a multi–processor machine and a multi–tape Turing machine). However, even if the two models of computation are found not to be equivalent, they still share some properties, since both of them model parallel computations. Then, an alternate question to the one above is: do they share the property of forming infinite hierarchies? That is, is there a (nontrivial) infinite hierarchy, this time with respect to the number of processors on a multi–processor abstract machine, which is similar in spirit to the hierarchy found in this paper for the real–time Turing machines?

References

[1] S. G. Akl, *Parallel Computation: Models and Methods*, Prentice-Hall, Upper Saddle River, NJ, 1997.

[2] ——, *Secure file transfer: A computational analog to the furniture moving paradigm*, Tech. Rep. 99-422, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1999. http://www.qucis.queensu.ca/~akl/techreports/furniture.ps.

[3] S. G. Akl and S. D. Bruda, *Real-time optimization: Beyond speedup*, to appear in: Parallel Processing Letters, 9 (1999). For a preliminary version see http://www.cs.queensu.ca/~akl/techreports/beyond.ps.

[4] ——, *Real-time cryptography: Beyond speedup II*, Tech. Rep. 99-423, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1999. http://www.cs.queensu.ca/~akl/techreports/realcrypto.ps.

[5] ——, *Real-time numerical computation: Beyond speedup III*, Tech. Rep. 99-424, Department of Computing and Information Science, Queen's University, Kingston, Ontario, Canada, 1999. http://www.cs.queensu.ca/~akl/techreports/realnum.ps.

[6] S. G. Akl. and L. Fava Lindon, *Paradigms admitting superunitary behaviour in parallel computation*, Parallel Algorithms and Applications, 11 (1997), pp. 129–153.

[7] R. V. Book and S. A. Greibach, *Quasy–realtime languages*, Mathematical Systems Theory, 4 (1970), pp. 97–111.

[8] S. D. Bruda and S. G. Akl, *On the data-accumulating paradigm*, in Proceedings of the Fourth International Conference on Computer Science and Informatics, Research Triangle Park, NC, October 1998, pp. 150–153. For a preliminary version see http://www.cs.queensu.ca/~bruda/www/data_accum.

[9] ——, *The characterization of data-accumulating algorithms*, in Proceedings of the International Parallel Processing Symposium, San Juan, Puerto Rico, 1999, pp. 2–6. For a preliminary version see http://www.cs.queensu.ca/~bruda/www/data_accum2.

[10] ——, *A case study in real-time parallel computation: Correcting algorithms*, in Proceedings of the Midwest Workshop on Parallel Processing, Kent, OH, 1999. For a preliminary version see http://www.cs.queensu.ca/~bruda/www/c-algorithms.

[11] P. C. Fischer, *Turing machines with a schedule to keep*, Information and control, 11 (1967), pp. 138–146.

[12] P. C. Fischer and C. M. R. Kintala, *Real-time computations with restricted nondeterminism*, Mathematical Systems Theory, 12 (1979), pp. 219–231.

[13] H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[14] F. Luccio and L. Pagli, *Computing with time–varying data: Sequential complexity and parallel speed–up*, Theory of Computing Systems, 31 (1998), pp. 5–26.

[15] M. O. Rabin, *Real time computations*, Israel Journal of Mathematics, 1 (1963), pp. 203–211.

[16] D. Rappaport, *Private communication.*

[17] A. L. Rosenberg, *Real-time definable languages*, Journal of the ACM, 14 (1967), pp. 645–662.

[18] ——, *On the independence of real-time definability and certain structural properties of context-free languages*, Journal of the ACM, 15 (1968), pp. 672–679.

[19] H. Yamada, *Real-time computation and recursive functions not real-time computable*, IRE Transactions on Electronic Computers, EC-11 (1962), pp. 753–760.