A NOTE ON ACCELERATED TURING MACHINES

CRISTIAN S. CALUDE, LUDWIG STAIGER

ABSTRACT. In this note we prove that any Turing machine which uses only a finite computational space for every input cannot solve an uncomputable problem even in case it runs in accelerated mode. We also propose two ways to define the language accepted by an accelerated Turing machine. Accordingly, the classes of languages accepted by accelerated Turing machines are the closure under Boolean operations of the sets Σ_1 and Σ_2.

1. ACCELERATED TURING MACHINES

"Acceleration" was first discussed by Weyl [30] in 1927 (and independently by Blake [4] and Russell [19]) in the form of the potential realisation of a process in which each step takes half of the time of the previous step. Copeland [8] and Stewart [26] applied this idea to Turing computations. An accelerated Turing machine (sometimes called Zeno machine) is a Turing machine that takes 2^{-n} units of time (say seconds) to perform its nth step; we assume that steps are in some sense identical except for the time taken for their execution. Such a machine can run an infinite number of steps in one unit of time. Accelerated Turing machines have been studied by various authors including Barrow [3], Boolos and Jeffrey [5], Calude and Păun [6], Ord [15], Potgieter [16], Shagrir [20, 21], Svozil [27].

The main feature of an accelerated Turing machine consists in its capability of computing in a finite time an infinite sequence of steps, thus allowing it to solve uncomputable problems. For example, the following (informal) accelerated Turing machine can solve the halting problem of an arbitrarily given Turing machine T and input w in finite time:

```
begin program
write 0 on the first position of the output tape;
set i = 1;
begin loop simulate the first i steps of T on w;
         if T(w) has halted, then write 1 on the
         first position of the output tape;
         i = i + 1;
 end loop
```

Calude: Department of Computer Science, The University of Auckland, Private Bag 92019, Auckland, New Zealand, cristian@cs.auckland.ac.nz.

Staiger: Martin-Luther-Universität Halle-Wittenberg, Institut für Informatik, D-06099 Halle, Germany, staiger@informatik.uni-halle.de.

Work done during L. Staiger's visit to CDMTCS in January 2009. An early version of this paper was presented at the *Workshop on Hypercomputation*, Ponta Delgada, Portugal, September 2009.

```
        end program
```

By inspecting the first position of the output tape we need one unit of time to run the above machine in order to decide whether $T(w)$ stops or not. Note that Svozil [27] proved that the halting problem of accelerated Turing machines is not decidable by any accelerated Turing machine. Relativistic computation offers a physical model for acceleration, see [12, 9, 1].

Are accelerated Turing machines physically possible? This is a challenging problem discussed by various authors, [14]. We contribute with a small result to this discussion by examining the computational space required by an (accelerated) Turing machine running an infinite computation: *is it finite or not?* This question was posed to the first author by Fearnley [11].

2. Is the space used by an accelerated Turing Machine always finite?

Let us start with the following informal example:

```
set i=0;
begin loop i=i+1;
end loop
```

It is clear that the accelerated Turing machine executing the above set of instructions needs an infinite computational space. Is this just an accident or do we have a more general situation?

Before being tempted by a hasty answer let us note that the following set of instructions computation is infinite, but requires only a finite amount of space:

```
set i=1;
while (i > 0) do i=1;
end while
```

To be able to answer the above question we fix a formal model of Turing machine and state a few general facts. We assume that reader is familiar with the basics of Turing computability, e.g. [22, 29].

Let $M = (X, \Gamma, S, s_0, s_a, \Box, \delta)$ be a Turing machine in which X is the input alphabet, $\Gamma \supset X$ is the working tape alphabet, S is the set of states, s_0 is the initial state, s_a is the accept state, $\Box \in \Gamma \setminus X$ is the blank symbol[1], and δ is the (partial) transition function. We assume that the Turing machine has one input read-only tape (where initially the input is written on) and k, $k \geq 1$ working tapes. If we need an output tape (for writing the results of computations) we use working tape k. The machine starts its processing in state s_0 by scanning the first symbol of the input word.

A configuration of the Turing machine with k working tapes on input x is a $2k+2$-tuple $(i, s, u_1, v_1, \ldots, u_k, v_k)$ where i, $0 \leq i \leq |x| + 1$ denotes the position of the head on the input

[1]We explicitly exclude the blank symbol from the input alphabet.

tape, s is the current state and $u_j \in \Gamma^*$ and $v_j \in \Gamma^*$, $u_j \notin \Box \cdot \Gamma^*, v_j \notin \Gamma^* \cdot \Box$ are the contents of the working tape j, $1 \leq j \leq k$ to the left or right, respectively, of the head position.

The successor configuration κ' of a configuration κ is derived as usual for multi-tape Turing machines (cf. [2, 29]).

The computation of M on x started in s_0 is a sequence of configurations starting with $\kappa_0 = (1, s_0, \varepsilon, \ldots, \varepsilon)$ each of which is a successor of its predecessor.

A word x is accepted by M if the computation of M started in s_0 on x stops in s_a. The language accepted by M is the set of words accepted by M.

Let $M = (X, \Gamma, S, s_0, s_a, \Box, \delta)$ be a Turing machine and x an input word. We define the computational space used by M on x, $space_M(x)$, to be the number—finite or infinite—of cells used by M during its computation on x (or, with input x); a cell used once is counted as used. Obviously, if $space_M(x)$ is finite then the computation process as described above can have only a finite number of different configurations. This observation will be crucial for our further considerations.

The function $time_M(x)$ denotes the number of steps executed by M on input x (see [2, 29]). By $M(x) < \infty$ we denote the fact that M stops on x. The reader is warned not to confuse our space function $space_M$ with the space complexity usually used in complexity theory (see [29]) which is defined by

$$s_M(x) = \begin{cases} space_M(x), & \text{if } M(x) < \infty, \\ \infty, & \text{otherwise}. \end{cases} \tag{1}$$

Clearly, $space_M(x) < \infty$ whenever $M(x) < \infty$, and $M(x) = \infty$ iff $time_M(x) = \infty$ iff $s_M(x) = \infty$.

The *halting problem for a particular Turing machine M* is the problem of deciding given x whether $M(x) < \infty$. It is well known that the halting problem for most Turing machines M is undecidable.

Following the argument of Lemma 2.25 in [2] one could prove that if, for a computable function $f : \mathbb{N} \to \mathbb{N}$, $space_M(x) \leq f(|x|)$ whenever M halts on x, then the halting problem for this particular machine M is decidable. We show that one can drop the requirement on a computable upper bound for $space_M$.

We start with a more general result.

Theorem 1. *There is a uniformly effective procedure which transforms every Turing machine M into a machine $\mathcal{D}_M$ which accepts the same inputs as M and has the property that $\mathcal{D}_M$ halts on all inputs x such that $space_M(x) < \infty$.*

Proof. The machine $\mathcal{D}_M$ works as follows. It runs the machine M on input x and simultaneously keeps track of a list of all configurations the machine M has run through. Then three cases are possible.

If M stops then $\mathcal{D}_M$ stops, too, and accepts x iff M accepts x.

As soon as one configuration appears twice in the list $\mathcal{D}_M$ stops and rejects the input.

If M does not stop and no configuration is repeated then $\mathcal{D}_M$ runs indefinitely.

To prove the assertion it suffices to remark that, since M is a deterministic machine, if $space_M(x) < \infty$ and the computation is infinite then necessarily one configuration is repeated and thus the sequence of configurations is eventually periodic; in particular, no new configuration will appear.

$\square$

The same idea can be used to prove the following result.

Theorem 2. *If for every x, $space_M(x) < \infty$, then the halting problem for M is decidable.*

Proof. Having in mind the proof of Theorem 1, one constructs an observer Turing machine $\mathcal{O}_M$ that lists all configurations of M generated by the computation $M(x)$ and continues as follows:

(1) If M stops on x then $\mathcal{O}_M$ stops too and declares $M(x) < \infty$.
(2) If M does not stop on x then on the first repetition in the list of configurations generated by $M(x)$ the machine $\mathcal{O}_M$ stops and declares that $M(x) = \infty$. $\square$

Corollary 3. *If the halting problem for M is undecidable then $\{x \in X^* : space_M(x) = \infty\} \neq \emptyset$.*

Corollary 4. *The set $\{(M, x) : M$ is a Turing machine, $x \in X^*$, $space_M(x) < \infty\}$ is computably enumerable but not computable.*

As Corollary 4 shows, our decidability result (Theorem 2) for Turing machines using only a finite amount of space does not allow to solve the general *halting problem*: given a pair (M, x), decide whether the machine M halts on x. Following a suggestion of one referee we mention that the following weaker versions of this problem are decidable.

Theorem 5. *Let $f : \mathbb{N} \to \mathbb{N}$ be a computable function. Then there is a Turing machine $\mathcal{D}$ which, given a pair (M, x), decides whether the machine M halts on x in space $space_M(x) \leq f(|x|)$.*

If, moreover, f is space constructible and $f(n) \geq \log_2 n$, then this decision procedure runs in space[2] *bounded by $space_{\mathcal{D}}(x) = s_{\mathcal{D}}(x) \leq f(|x|)$.*

Here, as usual, a function $f : \mathbb{N} \to \mathbb{N}$ is called *space constructible* if there is a Turing machine M_f which maps the binary expansion $\text{bin}(n)$ of n to the binary expansion of $f(n)$ using only space $s_{M_f}(\text{bin}(n)) \leq |\text{bin}(f(n))| \leq \log_2(f(n)) + 1$.

A Turing machine M running in 'accelerated mode' is denoted by A_M. In other words, M and A_M have the same description, but M runs in normal mode, i.e. each instruction is executed in a fixed unit of time, while A_M runs in an accelerated mode. Observe that $M(x) = \infty$ iff $time_M(x) = \infty$ iff $time_{A_M}(x) = 1$. The function $time_M$ classically counts the number of steps executed by M, while $time_{A_M}$ measures the length of a time interval; with the assumption that each step takes precisely one unit of time, these functions become essentially equivalent.

[2]For the function $s_{\mathcal{D}}$ see Eq. (1).

There is a similarity between computational time and space; however, this parallel is not perfect. For example, it is not true that an accelerated Turing machine which uses unbounded space has to use an infinite space for some input (as it seems to be claimed in Ord [15, p. 24]). The reason is that the space used by the machine on every input x can be finite, although it grows indefinitely with $|x|$.

Let $\chi_M : X^* \to \{0,1\}$ be the function defined by

$$\chi_M(x) = \begin{cases} 1, & \text{if } M(x) < \infty, \\ 0, & \text{otherwise.} \end{cases}$$

This function can always be computed by an accelerated Turing machine $A_{M'}$ in finite time.[3] If the computational space is finite for every input, then acceleration does not add computational power:

Corollary 6. *Let A_M be an accelerated Turing machine with $\text{space}_{A_M}(x) < \infty$ for all inputs x. Then the function χ_M is Turing computable. The Turing machine computing χ_M is not necessarily M.*

3. Computational power

How can we use accelerated Turing machines to trespass the Turing barrier, more precisely, to accept languages other than computably enumerable ones? A proposal based on physical considerations to use accelerated Turing machines with an oracle provided by another accelerated Turing machine was made in [31]. Here we pursue a different approach dating back to the late 1970s where infinite acceptance processes for Turing machines were considered [7, 10, 25].

These processes consider acceptance conditions based on the set of states occurring or occurring infinitely often during the computation process. To this end we pair the machine M with one or two observer machines M' and M''. There are two ways to observe the computation of M and, consequently, decide its output.

In the first case the output is based on the set of states occurring during the computation. The machine M' simply collects the (finite) set of states $\mathcal{S}_x$ occurring during M's computation process on input x.

In the second case the output is based on the set of states occurring infinitely often during the computation. During the computation of M on x the first observer machine M' writes into cell i of its output tape successively (a symbol denoting) the set of states $\mathcal{S}_x(i,t)$ the machine M runs through starting from step i up to step t. Thus, after finishing its work, cell i contains (a symbol denoting) the set of states M has run through starting from moment i on. This sequence of sets is non-increasing, so the second observer machine M'' can compute its limit $\mathcal{S}_x$.

In both cases, the input word x is accepted according to whether $\mathcal{S}_x$ satisfies a previously given condition described below.

[3] $A_{M'}$ is not necessarily equal to A_M.

The processes considered here may stop or not after finitely many steps. To treat both cases in a uniform way we assume in the first case that the last state is repeated indefinitely. In this way we don't need to test whether the computation of M eventually stops or not, so we avoid paradoxes like the Thompson lamp [28].

A detailed account of such acceptance processes is given in the survey papers [13, 24]. We denote by $ran(M,x)$ ($in(M,x)$, respectively) the set of states $\mathcal{S}_x$ of M occurring (occurring infinitely often, respectively) in the computation process on input x. For an accelerated Turing machine $M = (X, \Gamma, S, s_0, s_a, \Box, \delta)$ and a subset $\mathcal{T} \subseteq 2^S$ define the following languages

$$\mathrm{AT}_{ran}(M,\mathcal{S}) = \{x : ran(M,x) \in \mathcal{T}\} \tag{2}$$

$$\mathrm{AT}_{in}(M,\mathcal{S}) = \{x : in(M,x) \in \mathcal{T}\} \tag{3}$$

Let Σ_1, Π_1, Π_2 and Σ_2 be the first classes of the arithmetical hierarchy of languages (see [17, 29]). In particular, Σ_1 is the class of computably enumerable languages and Π_1 is the class of their complements. By $\mathsf{Bool}(\mathcal{M})$ we denote the closure of a set of sets $\mathcal{M}$ under Boolean operations.

Based on [23], we have the following results:

Theorem 7. *For the classes of accepted languages the following identities hold true:*

$$\begin{aligned}\{\mathrm{AT}_{ran}(M,\mathcal{S}) : M = (X,\Gamma,S,s_0,\Box,\delta) \text{ an ATM }\} &= \mathsf{Bool}(\Sigma_1),\\ \{\mathrm{AT}_{in}(M,\mathcal{S}) : M = (X,\Gamma,S,s_0,\Box,\delta) \text{ an ATM }\} &= \mathsf{Bool}(\Sigma_2).\end{aligned}$$

Acknowledgment

We thank L. Fearnley for posing the problem discussed in this note, P. Potgieter for illuminating discussions, and the anonymous referees for excellent critical comments.

References

[1] H. Andréka, I. Németi, P. Németi, J. X. Madarász, G. Székely. *Logic and Relativity Theory*, Course Notes 2006, 27 pages.
[2] J. L. Balcázar, J. Díaz, L. Gabarró, *Structural Complexity* I, Second revised ed., Springer-Verlag, Berlin 1995.
[3] J. Barrow. *The Infinite Book. A Short Guide to the Boundless, Timeless and the Endless*, Jonathan Cape, London, 2005.
[4] R. M. Blake. The paradox of temporal process, *J. Philos.* 23 (1926), 645–654.
[5] G. Boolos, R. C. Jeffrey. *Computability and Logic*, Cambridge University Press, Cambridge, 1980.
[6] C. S. Calude, G. Păun. Bio-steps beyond Turing, *Biosystems* 77 (1–3) (2004), 175–194.
[7] R. S. Cohen and A. Y. Gold. ω-computations on Turing machines, *Theoret. Comput. Sci.* 6 (1978), 1–23.
[8] B. Copeland. Accelerating Turing machines, *Minds and Machines* 12 (2) (2002), 281–300.
[9] G. Etesi, I. Németi. Non-Turing computations via Malament-Hogarth space-times, *International Journal of Theoretical Physics* 41 (2002), 341–370.
[10] L. H. Landweber. Decision problems for ω-automata, *Math. Syst. Theory* 3, 4 (1969), 376–384.
[11] L. Fearnley. Email to (and discussions with) C. Calude, 3 December 2008.

[12] M. Hogarth. Does general relativity allow an observer to view eternity in a finite time? *Foundations of Physics Letters* 5 (1992), 173–181.

[13] J. Engelfriet, H. J. Hoogeboom. X-automata on ω-words, *Theoret. Comput. Sci.* 110, 1 (1993), 1–51.

[14] L. Floridi (ed.). *The Blackwell Guide to the Philosophy of Computing and Information*, Blackwell, Malden MA, 2004.

[15] T. Ord. *Hypercomputation: Computing More than the Turing Machine*, Honours Thesis, Computer Science Department, University of Melbourne, Australia, 2002; `arxiv.org/ftp/math/papers/0209/0209332.pdf`.

[16] P. H. Potgieter. Zeno machines and hypercomputation, *Theoretical Computer Science* 358 (2006), 23–33. Also in `arXiv:cs.CC/0412022`.

[17] H. Rogers. *Theory of Recursive Functions and Effective Computability*, McGraw Hill, New York 1967.

[18] G. Rozenberg, A. Salomaa (eds.). *Handbook of Formal Languages*, Springer-Verlag, Berlin, 1997.

[19] B Russell. The limits of empiricism, *Proc. Aristotelian Soc.* 36 (1936), 131–150.

[20] O. Shagrir. Accelerating Turing machines, in F. Stadler and M. Stroltzner (eds.). *Time and History (Papers of the 28th International Wittgenstein Symposium)*, Austrian Ludwig Wittgenstein Society, 2005, pp. 276–278.

[21] O. Shagrir. Super-tasks, accelerating Turing machines and uncomputability, *Theoretical Computer Science*, 317 (2004), 105–114.

[22] M. Sipser. *Introduction to the Theory of Computation*, PWS, Boston, 2006, second edition.

[23] L. Staiger, ω-computations on Turing machines and the accepted languages, in L. Lovász and E. Szemerédi (eds.). *Coll. Math. Soc. Janos Bolyai* No.44, North Holland, Amsterdam 1986, 393–403.

[24] L. Staiger. ω-languages, in [18], Vol. 3, 339–387.

[25] L. Staiger and K. Wagner. Rekursive Folgenmengen I, *Zeitschr. Math. Logik u. Grundl. Mathematik* 24, 6 (1978), 523–538 (in German). A preliminary version appeared as: K. Wagner and L. Staiger, Recursive ω-languages, in M. Karpiński (ed.). *Proc. Fundamentals of Computation Theory '77*, Lect. Notes Comput. Sci. 56, Springer Verlag, Berlin 1977, 532–537.

[26] I. Stewart. Deciding the undecidable, *Nature* 352 (1991), 664–665.

[27] K. Svozil. The Church-Turing thesis as a guiding principle for physics, in: C. Calude, J. Casti, M. Dinneen (eds.). *Unconventional Models of Computation*, Springer, Berlin, 1998, 371–385.

[28] K. Svozil. On the Brightness of the Thomson Lamp. A Prolegomenon to Quantum Recursion Theory, *CDMTCS Research Report* 360, 2009.

[29] K. Wagner, G. Wechsung. *Computational Complexity*, Deutscher Verlag der Wissenschaften, Berlin, 1986.

[30] H. Weyl. *Philosophy of Mathematics and Natural Science*, Princeton University Press, Princeton, 1949.

[31] J. Wiedermann, J. van Leeuwen. Relativistic computers and non-uniform complexity theory, in C. S. Calude, M. J. Dinneen, F. Peper (eds.). *Unconventional Models of Computation*, Lecture Notes Comput. Sci. 2509, Springer-Verlag, Heidelberg, 2002, 278–298.

A Note on Accelerated Turing Machines

C. S. Calude[1] and L. Staiger[2]
[1]University of Auckland, NZ
[2]Martin-Luther-Universität, Germany

CDMTCS-350
February 2009