Math. Struct. in Comp. Science (2010), *vol.* 21, *pp.* 1–7.
doi:10.1017/S0960129510000344

A note on accelerated Turing machines

CRISTIAN S. CALUDE† and LUDWIG STAIGER‡

†*Department of Computer Science, The University of Auckland,*
Private Bag 92019, Auckland, New Zealand
Email: cristian@cs.auckland.ac.nz
‡*Martin-Luther-Universität Halle-Wittenberg,*
Institut für Informatik, D - 06099 Halle, Germany
Email: staiger@informatik.uni-halle.de

Received 22 April 2009; revised 10 June 2010

In this paper we prove that any Turing machine that uses only a finite computational space for every input cannot solve an uncomputable problem even when 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 (Weyl 1949) in 1927 (and independently by Blake (Blake 1926) and Russell (Russell 1936)) in the form of the potential realisation of a process in which each step takes half of the time of the previous step. Copeland (2002) and Stewart (1991) applied this idea to Turing computations. An accelerated Turing machine (sometimes called a 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 (Barrow 2005), Boolos and Jeffrey (Boolos and Jeffrey 1980), Calude and Păun (Calude and Păun 2004), Ord (Ord 2002), Potgieter (Potgieter 2006), Shagrir (Shagrir 2005; 2004) and Svozil (Svozil 1998).

The main feature of an accelerated Turing machine is its ability to compute an infinite sequence of steps in a finite time, 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
 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 (Svozil 1998) proved that the halting problem for accelerated Turing machines is not decidable by any accelerated Turing machine. Relativistic computation offers a physical model for acceleration (Hogarth 1992; Etesi and Németi 2002; Andréka *et al.* 2006).

But are accelerated Turing machines physically possible? This is a challenging problem discussed by various authors (Floridi 2004). In this paper we contribute 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 by Fearnley to the first author (Fearnley 2008).

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 does it indicate a more general situation?

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

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

In order to answer the above question, we fix a formal model of a Turing machine and state a few general facts. We assume familiarity with the basics of Turing computability as in, for example, Sipser (2006) and Wagner and Wechsung (1986).

Let $M = (X, \Gamma, S, s_0, s_a, \square, \delta)$ 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, $\square \in \Gamma \setminus X$ is the blank symbol[†] and δ is the (partial) transition function. We assume that the Turing machine has one input read-only tape (on which the input has initially been written) and k, $k \geqslant 1$ 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, $0 \leqslant i \leqslant |x|+1$ denotes the position of the head on the input tape, s is the current state and $u_j \in \Gamma^*$ and $v_j \in \Gamma^*$, $u_j \notin \square \cdot \Gamma^*, v_j \notin \Gamma^* \cdot \square$ are the contents of the working tape j, $1 \leqslant j \leqslant k$ to the left or right, respectively, of the head position.

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

The successor configuration κ' of a configuration κ is derived in the usual way for multi-tape Turing machines (*cf.* Balcázar *et al.* (1995) and Wagner and Wechsung (1986)).

The computation of M on x started in s_0 is a sequence of configurations starting with $\kappa_0 = (1, s_0, \varepsilon, \dots, \varepsilon)$, 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 be an input word. We define the computational space used by M on x, $space_M(x)$, to be the (finite or infinite) number 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, 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 Balcázar *et al.* (1995) and Wagner and Wechsung (1986)). We use $M(x) < \infty$ to denote the fact that M stops on x. Care should be taken not to confuse our space function $space_M$ with the space complexity usually used in complexity theory (Wagner and Wechsung 1986), 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 $M(x) = \infty$ if and only if $time_M(x) = \infty$ if and only if $s_M(x) = \infty$.

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 Balcázar *et al.* (1995, Lemma 2.25), one could prove that if for a computable function $f : \mathbb{N} \to \mathbb{N}$ we have $space_M(x) \leqslant f(|x|)$ whenever M halts on x, then the halting problem for this particular machine M is decidable. We show that the computable upper bound for $space_M$ requirement can be dropped.

However, we start with a more general result.

Theorem 1. There is a uniformly effective procedure that transforms every Turing machine M into a machine $\mathcal{D}_M$ that 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.

Three cases are possible:

(1) If M stops, then $\mathcal{D}_M$ stops too, and accepts x if and only if M accepts x.
(2) As soon as one configuration appears twice in the list, $\mathcal{D}_M$ stops and rejects the input.
(3) If M does not stop and no configuration is repeated, then $\mathcal{D}_M$ runs indefinitely.

To prove the assertion, it suffices to note 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. □

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. Bearing in mind the proof of Theorem 1, we construct 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 $M(x) = \infty$. □

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

Corollary 4. The set $\{(M, x) : M \text{ is a Turing machine}, x \in X^*, space_M(x) < \infty\}$ 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 us to solve the general *halting problem*: given a pair (M, x), decide whether the machine M halts on x. Following a suggestion of one of the referees, 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 $\mathcal{D}$ that, given a pair (M, x), decides whether the machine M halts on x in space $space_M(x) \leqslant f(|x|)$.

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

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

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, that is, each instruction is executed in a fixed unit of time, while A_M runs in an accelerated mode. Observe that $M(x) = \infty$ if and only if $time_M(x) = \infty$ if and only if $time_{A_M}(x) = 1$. 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.

There is a similarity between computational time and space, but this parallel is not perfect. For example, it is not true that an accelerated Turing machine that uses unbounded space has to use an infinite amount of space for some input (as appears to be

[†] See Equation (1) for the function $s_{\mathcal{D}}$.

claimed in Ord (2002, page 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[†]. 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 $\mathit{space}_{A_M}(x) < \infty$ 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 cross 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 (Wiedermann and van Leeuwen 2002). Here we pursue a different approach dating back to the late 1970s in which infinite acceptance processes for Turing machines were considered (Cohen and Gold 1978; Landweber 1969; Staiger and Wagner 1977).

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:

(1) 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.

(2) 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, which is described below.

The processes considered here may or may not stop after finitely many steps. To treat both cases in a uniform way, we assume in the first case that the last state is

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

repeated indefinitely. In this way, we do not need to test whether the computation of M eventually stops or not, so we avoid paradoxes like the Thompson lamp (Svozil 2009).

A detailed account of such acceptance processes is given in the survey papers Engelfriet and Hoogeboom (1993) and Staiger (1997). We use $ran(M,x)$ and $in(M,x)$ to denote the set of states $\mathcal{S}_x$ of M occurring and occurring infinitely often, respectively, in the computation process on input x. For an accelerated Turing machine $M = (X, \Gamma, S, s_0, s_a, \square, \delta)$ and a subset $\mathcal{T} \subseteq 2^S$, we 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 (Rogers 1967; Wagner and Wechsung 1986). In particular, Σ_1 is the class of computably enumerable languages and Π_1 is the class of their complements. We use $\mathsf{Bool}(\mathcal{M})$ to denote the closure of a set of sets $\mathcal{M}$ under Boolean operations.

From Staiger (1986), we have the following results.

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

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

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. This work was 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 in September 2009.

References

Andréka, H., Németi, I., Németi, P., Madarász, J. X. and Székely, G. (2006) Logic and Relativity Theory, Course Notes.

Balcázar, J. L., Díaz, J. and Gabarró, L. (1995) *Structural Complexity I*, Second revised edition, Springer-Verlag.

Barrow, J. (2005) *The Infinite Book. A Short Guide to the Boundless, Timeless and the Endless*, Jonathan Cape.

Blake, R. M. (1926) The paradox of temporal process. *J. Philos.* **23** 645–654.

Boolos, G. and Jeffrey, R. C. (1980) *Computability and Logic*, Cambridge University Press.

Calude, C. S. and Păun, G. (2004) Bio-steps beyond Turing. *Biosystems* **77** (1-3) 175–194.

Cohen, R. S. and Gold, A. Y. (1978) ω-computations on Turing machines. *Theoret. Comput. Sci.* **6** 1–23.

Copeland, B. (2002) Accelerating Turing machines. *Minds and Machines* **12** (2) 281–300.

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

Etesi, G. and Németi, I. (2002) Non-Turing computations via Malament–Hogarth space-times. *International Journal of Theoretical Physics* **41** 341–370.

Fearnley, L. (2008) Email to (and discussions with) C. Calude, 3 December 2008.

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

Floridi, L. (ed.) (2004) *The Blackwell Guide to the Philosophy of Computing and Information*, Blackwell.

Landweber, L. H. (1969) Decision problems for ω-automata. *Math. Syst. Theory* **3** (4) 376–384.

Ord, T. (2002) Hypercomputation: Computing More than the Turing Machine. Honours Thesis, Computer Science Department, University of Melbourne (Available at `arxiv.org/ftp/math/papers/0209/0209332.pdf`.)

Potgieter, P. H. (2006) Zeno machines and hypercomputation. *Theoretical Computer Science* **358** 23–33. (Available at `arXiv:cs.CC/0412022`.)

Rogers, H. (1967) *Theory of Recursive Functions and Effective Computability*, McGraw Hill.

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

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

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

Sipser, M. (2006) *Introduction to the Theory of Computation*, second edition, PWS .

Staiger, L. (1986) ω-computations on Turing machines and the accepted languages. In: Lovász, L. and Szemerédi, E. (eds.) *Coll. Math. Soc. Janos Bolyai* **44** 393–403.

Staiger, L. (1997) ω-languages. In: Rozenberg, G. and Salomaa, A. (eds.) (1997) *Handbook of Formal Languages* **3**, Springer-Verlag 339–387.

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

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

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

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

Wagner, K. and Wechsung, G. (1986) *Computational Complexity*, Deutscher Verlag der Wissenschaften.

Weyl, H. (1949) *Philosophy of Mathematics and Natural Science*, Princeton University Press.

Wiedermann, J. and van Leeuwen, J. (2002) Relativistic computers and non-uniform complexity theory. In: Calude, C. S., Dinneen, M. J. and Peper, F. (eds.) Unconventional Models of Computation. *Springer-Verlag Lecture Notes in Computer Science* **2509** 278–298.