Accelerated Turing Machines

Computational Space and Power

Cristian S. Calude, Ludwig Staiger

UC2009 Hypercomputation Workshop
September 2009

Outline

- Accelerated computations
- Is the space used by an ATM always finite?
- Computational power of ATMs

Basic idea

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.

Basic idea

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.

Basic idea

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 discovered by Weyl in 1927 and studied by various authors including Barrow, Boolos and Jeffrey, Calude and Păun, Copeland, Ord, Potgieter, Shagrir, Stewart, Svozil .

Solving the halting problem 1

The following ATM can solve the halting problem of an arbitrarily given TM 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
```

Solving the halting problem 2

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.

Solving the halting problem 2

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.

Alternatively, we can clock the computation and see whether the machine itself has halted before one unit of time or not.

Are accelerated Turing machines physically possible?

This challenging problem was discussed by various authors.
Relativistic computation offers a physical model for acceleration.

Are accelerated Turing machines physically possible?

This challenging problem was discussed by various authors. Relativistic computation offers a physical model for acceleration.

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?*

- Accelerated computations
- Is the space used by an ATM always finite?
- Computational power of ATMs

An informal example

The TM executing the above set of instructions needs an infinite computational space:

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

Is this just an accident or do we have a more general situation?

A second example

The following TM executes an infinite computation with only a finite amount of space:

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

A formal model of TM 1

Let $(X, \Gamma, S, s_0, \square, \delta)$ be a Turing machine in which X is the input alphabet, $\Gamma \supset X$ is the work tape alphabet, S is the set of states, s_0 is the initial 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 tape where initially the input is written on, and the machine starts its processing in state s_0 by scanning the first symbol of the input word.

A formal model of TM 2

Let $M = (X, \Gamma, S, s_0, \Box, \delta)$ 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 on x.

The function $time_M(x)$ denotes the number of steps executed by M on input x. By $M(x) < \infty$ we denote the fact that M stops on x.

Clearly, $space_M(x) < \infty$ whenever $M(x) < \infty$, and

$$M(x) = \infty \text{ iff } time_M(x) = \infty.$$

A condition for a computation to halt

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

A condition for a computation to halt

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

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

A condition for a computation to halt

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

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

Corollary. *If $space_M(x) < \infty$, then the problem whether $M(x)$ halts or not is algorithmically decidable.*

A condition for a computation to halt

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

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

Corollary. *If $space_M(x) < \infty$, then the problem whether $M(x)$ halts or not is algorithmically decidable.*

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

Computational time and space

There is a similarity between computational time and space; however, this parallel is not perfect.

Computational time and space

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's Thesis). The reason is that every reasonable computable problem requires at least the input data x to be scanned, so it needs at least a space greater than the length of x.

Main result

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

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

can always be computed by an ATM $A_{M'}$—not necessarily equal to A_M—in finite time.

Main result

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

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

can always be computed by an ATM $A_{M'}$—not necessarily equal to A_M—in finite time.

If the computational space is finite for every input, then acceleration does not add computational power:

Theorem. *Let A_M be an ATM with $\text{space}_{A_M}(x) < \infty$ for all inputs x. Then the function χ_M is Turing computable (not necessarily by M).*

- Accelerated computations
- Is the space used by an ATM always finite?
- Computational power of ATMs

ATMs with oracles

How can we use ATMs to trespass the Turing barrier, more precisely, to accept languages other than computably enumerable ones?

ATMs with oracles

How can we use ATMs to trespass the Turing barrier, more precisely, to accept languages other than computably enumerable ones?

A proposal based on the idea to use ATMs with an oracle provided by another ATM was made by Wiedermann and van Leeuwen (2002).

Acceptance based on states 1

We consider acceptance conditions based on the set of states occurring or occurring infinitely often during the computation process.

Acceptance based on states 1

We 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.

Acceptance based on states 1

We 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.

Acceptance based on states 2

In the second case the output is based on the set of states occurring occurring infinitely often during the computation.

Acceptance based on states 2

In the second case the output is based on the set of states occurring 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.

Acceptance based on states 2

In the second case the output is based on the set of states occurring 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 $\mathcal{S}_x$.

Acceptance based on states 3

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.

Acceptance based on states 4

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 ATM M and a subset $\mathcal{S} \subseteq 2^S$ define the following languages

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

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

Let Σ_1, Π_1, Π_2 and Σ_2 be the first classes of the arithmetical hierarchy of languages.
By Bool($\mathcal{M}$) we denote the closure of a set of sets $\mathcal{M}$ under Boolean operations.

Acceptance based on states 5

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

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

Outline

- Accelerated computations
- Is the space used by an ATM always finite?
- Computational power of ATMs