Lossy counter machines and timed automata

Sławomir Lasota

Course notes

November 23, 2024

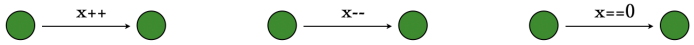
Contents

1
Lossy counter machines

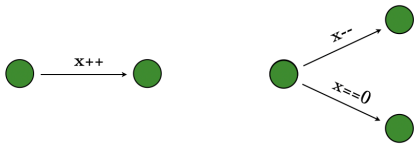
We start by recalling the model of deterministic *counter machines* (called also *Minsky machines*[1]).

Counter machines are finite-state machines that manipulate *counters* – variables that store nonnegative integers. Transitions of a counter machine, besides changing control states, perform a specified operation on a counter: increment by one, decrement by one, or zero test:

[1] Marvin Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961



Increment of a counter is executable unconditionally, decrement is only executable when the counter has positive value, and zero test is only executable when the counter is zero. There is one distinguished *initial* state and one *halting* state. A machine is *deterministic* if every control state (except for the halting one which has no outgoing transitions) has either exactly one outgoing transition which increments a counter, or exactly two outgoing transitions which decrement and zero test the same counter, respectively:



Formally, a configuration of a counter machine with d counters consists of a control state q and a vector of counter values $\vec{v} \in \mathbb{N}^d$. The initial configuration is the initial state with all counters equal to zero. A halting configuration is any configuration with the halting state. A *run* of a machine is a sequence $c_0, c_1, c_2, \ldots$ of configurations such that

1. c_0 is initial,
2. every next configuration c_{i+1} is obtained from c_i by applying a transition of the machine (which we denote as $c_i \longrightarrow c_{i+1}$),
3. the sequence is maximal, i.e., either ends in the halting configuration or is infinite.

Note that the successor configuration c_{i+1} in point 2. is determined uniquely by c_i. Therefore a deterministic counter machine has exactly one run, which may be either infinite (in which case the machine loops), or finite (in which case the run necessarily ends in the halting state – we call this run *halting* as well).

The model of (deterministic) counter machines is Turing-complete, and therefore the following *halting problem* is undecidable (even for machines with 2 counters):

Input: A deterministic counter machine $\mathcal{M}$.
Question: Is the only run of $\mathcal{M}$ halting?

In this chapter we focus on the following weakening of the model of deterministic counter machines: in each configuration along a run, every counter may spontaneously decrease by an arbitrary amount.

This model we call *lossy counter machines*.[2] Let Q be the set of control states of a lossy counter machine. We naturally extend the pointwise ordering $\sqsubseteq$ of vectors in $\mathbb{N}^d$, where d is the number of counters, to the configurations $Q \times \mathbb{N}^d$,

$$(q, \vec{v}) \sqsubseteq (q', \vec{v}') \text{ if } q = q' \text{ and } \vec{v} \sqsubseteq \vec{v}'.$$

[2] The model has been introduced in this paper:

Richard Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, 2003

By a run we mean now any maximal sequence $c_0, c_1, c_2, \ldots$ of configurations such that c_0 is initial, and

2′. every next configuration $c_{i+1} \sqsubseteq c'$ where $c_i \longrightarrow c'$.

Note that the lossy model stops being deterministic: a lossy machine may have many runs starting from the initial configuration, as a configuration may have many different successors. Therefore we need to reformulate the halting problem accordingly:

Input: A deterministic counter machine $\mathcal{M}$.
Question: Does $\mathcal{M}$ have a halting run?

This central problem turns out, quite surprisingly, decidable for the lossy model.

Theorem 1. The halting problem is decidable for lossy counter machines.

Also surprisingly, its computational complexity is tremendously high, namely it is not bounded by a *primitive recursive function*.[3] Most of computable functions $\mathbb{N} \to \mathbb{N}$ that appear in computer science are primitive recursive, for instance so are the following functions:

$$n^3 \qquad 2^n \qquad 2^{2^{2^n}} \qquad \text{tower}(n) = \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n \text{ times}}$$

We have the following lower complexity bound for the halting problem:

[3] Primitive recursive functions is the smallest subclass of recursive functions which includes some basic functions like successor or projections, and is closed under composition and under the scheme of primitive recursion:

$$h(0, y_1, \ldots, y_m) = f(y_1, \ldots, y_m)$$
$$h(x+1, y_1, \ldots, y_m) = g(h(x, y_1, \ldots, y_m), x, y_1, \ldots, y_m).$$

Theorem 2. The complexity of the halting problem for lossy counter machines in non-primitive recursive.

As we know that *Ackermann function*[4] is dominated by no primitive recursive function, it is enough to show that lossy counter machines can faithfully simulate computations of ordinary machines where the counters are bounded by the Ackermann function of the size of a machine.

The two theorems are proved in Sections 1.1 and 1.2, respectively.

[4] A non-primitive recursive (but still recursive) function. Using an auxiliary two argument function

$$A(0,n) = 2n$$
$$A(m+1,0) = 1$$
$$A(m+1,n+1) = A(m,A(m+1,n)),$$

we define the Ackermann function using diagonalisation:

$$A(n) = A(n,n).$$

REMARK 1. One can apply the analogous lossy relaxation also to other models. One relevant example is *lossy FIFO automata* (cf. Problem 14). The model is suitable for modeling finite-state systems that asynchronously exchange messages via unreliable communication channels.

1.1 *Decidability of the halting problem*

The halting problem is easily semi-decidable: the semi-decision procedure systematically enumerates all finite prefixes of runs, and terminates if some of them is a halting run. It is thus enough to demonstrate that non-halting is also semi-decidable. To this end, the semi-decision procedure will systematically enumerate all *non-halting witnesses*, to be defined below. To define them, we need a notion of *well quasi-order*.

A quasi-order[5] $(X, \preceq)$ we call *well quasi-order* (WQO) if every infinite sequence of elements of X

$$x_1,\ x_2,\ x_3,\ \ldots$$

admits a domination pair, i.e., a pair of elements $x_i \preceq x_j$ for some $i < j$. Observe that, in consequence of the definition, a WQO has no infinite antichain. Alternatively, a quasi order is a WQO if, and only if, it is well-founded and has only finite antichains (cf. Problem 1).

[5] The notion is traditionally defined for *quasi*-orders, while we will only use it for *partial* orders.

LEMMA 1 (Dickson). The set of vectors $\mathbb{N}^d$ of fixed length d, ordered pointwise, is a WQO.

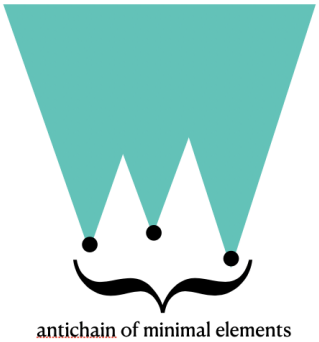
Let Q be the set of control states of a lossy counter machine. We deduce that the extension $\sqsubseteq$ of the pointwise ordering of vectors to the configurations $Q \times \mathbb{N}^d$, where d is the number of counters, is a WQO too:

COROLLARY 1. The set of configurations $Q \times \mathbb{N}^d$ of a counter machine, ordered by $\sqsubseteq$, is a WQO.

A subset $Y \subseteq Q \times \mathbb{N}^d$ is *downward closed* if whenever $y \in Y$ and $z \preceq y$ then $z \in Y$. Likewise we define *upward closed*[6] subsets $Y \subseteq Q \times \mathbb{N}^d$. We observe that the set of reachable configurations of a lossy counter machine, i.e., the set of all configurations appearing in some run, is downward closed.

In consequence, the set of all non-reachable configurations is upward closed. And, since the ordering $\sqsubseteq$ is a WQO, we have the following simple but very useful fact:

[6] An upward-closed subset of a WQO looks like this:



Lemma 2. Every upward closed subset $Y \subseteq Q \times \mathbb{N}^d$ is unambiguously represented by the set $\min Y$ of its minimal elements with respect to $\sqsubseteq$, and $\min Y$ is finite.

Indeed, the set $M = \min Y$ is finite, being an antichain, and Y is exactly the upward closure of M, i.e., $Y = {\uparrow}M = \left\{ y \in Q \times \mathbb{N}^d \mid \exists m \in M \,.\, m \sqsubseteq y \right\}$.

So prepared, we are now ready to define finite non-halting witnesses. As such a witness we choose any finite antichain $M \subseteq Q \times \mathbb{N}^d$ whose upper closure ${\uparrow}M$ satisfies the following conditions:

1. All halting configurations belong to ${\uparrow}M$.
2. The initial configuration does not belong to ${\uparrow}M$.
3. ${\uparrow}M$ is closed under the reverse of $\longrightarrow$: if $c \in {\uparrow}M$ and $c' \longrightarrow c$ then $c' \in {\uparrow}M$.

Correctness of the negative semi-decision procedure follows from the following fact:

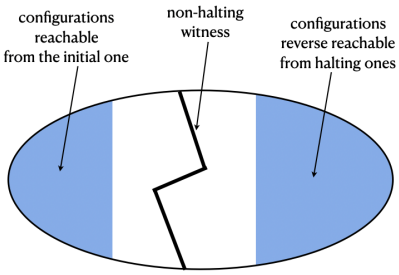
Lemma 3. The machine does not halt if, and only if, the machine admits a finite non-halting witness as defined above.

Proof. Denote by R the set of all configurations from which a halting one is reachable. In other words, these are configurations *reverse reachable* from halting ones. For the *only if* direction observe that the set R is upward closed and $M = \min R$ satisfies the defining conditions of non-halting witness. For the *if* direction assume a non-halting witness M. The upward closure ${\uparrow}M$ *excludes* the initial configuration and, due to the closure under the reverse of $\longrightarrow$, ${\uparrow}M$ *includes* R. Hence the initial configuration does not belong to R, i..e, the machine does not halt. □

Finally, we observe that this characterization yields a negative semi-decision procedure, as it can be effectively checked if a given finite subset $M \subseteq Q \times \mathbb{N}^d$ is a non-halting witness, i.e., if it is an antichain and satisfies the conditions 1–3 above (cf. Problem 3). In particular, condition 1 holds if, and only if, the configuration $(q^{\text{halt}}, \vec{0})$ belongs to M, where q^{halt} is the halting state.

Remark 2. There may be many different non-halting witnesses M that induce different upward closures ${\uparrow}M$. The least possible upward closure ${\uparrow}M$ is ${\uparrow}M = R$ (the set of all configurations reverse reachable from the halting ones). The largest possible upward closure contains all configurations non-reachable from the initial one.

Any witness can be seen a separator between configurations reachable from the initial one and configurations reverse reachable from the final ones, as shown on the figure. In general there is no guarantee the the negative semi-decision procedure finds any of these two extremal ones.



Furthermore, there is no algorithm to check if a given non-halting witness is the largest one. In other words, one can not compute a representation of the set of all non-reachable configurations (in consequence, one can not compute the set of all reachable configurations). On the

other hand, the set of configurations reverse reachable from the final ones can be computed (cf. Problem 7).

1.2 *Non-primitive recursive lower bound*

For the lower bound of Theorem 2 we design an algorithm to correctly compute Ackermann function using a lossy counter machine. The presentation is based on this[7] article but, for convenience, we prefer work with the variant of this fast-growing function defined above, different than the one used in the article. All functions below are from $\mathbb{N}$ to $\mathbb{N}$:

[7] Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. MFCS 2010*, volume 6281 of *LNCS*, pages 616–628. Springer, 2010

$$\begin{aligned} \mathrm{A}_0(n) &= 2n \\ \mathrm{A}_{k+1}(n) &= \mathrm{A}_k{}^n(1) = \underbrace{\mathrm{A}_k(\mathrm{A}_k(\ldots \mathrm{A}_k(1)\ldots))}_{n \text{ times}} \\ \mathrm{A}(n) &= \mathrm{A}_n(n). \end{aligned}$$

Note that all the auxiliary functions A_k are primitive recursive (why?). In particular: $\mathrm{A}_1(n) = 2^n$; the next function $\mathrm{A}_2(n) = \mathrm{tower}(n)$ composes n times A_1, starting from 1; the function $\mathrm{A}_3(n)$ is the supertower function that composes n times the tower function, starting from 1, and so on.

We use the following vectorial notation, which can be thought of as a configuration of a counter machine using $m+2$ counters:

$$\langle a_m, a_{m-1}, \ldots, a_0 \,;\, n \rangle = \mathrm{A}_m^{a_m}(\mathrm{A}_{m-1}^{a_{m-1}}(\ldots \mathrm{A}_1^{a_1}(\mathrm{A}_0^{a_0}(n))\ldots)).$$

When using this notation we implicitly assume that $a_m, a_{m-1}, \ldots, a_1, a_0 \in \mathbb{N}$ are nonnegative, but $n \in \mathbb{N} \setminus \{0\}$ is positive. In particular, the notation $\langle 1, 0, \ldots, 0 \,;\, n \rangle$ represents, intuitively, a request to compute $\mathrm{A}_m(n)$: The right-most counter will play a special role as it collects the result of the computation. The remaining $m+1$ counters we call *normal* counters, and the last (right-most) one we call *special* counter.

Towards computing the Ackermann function we observe the following equalities:

$$\begin{aligned} \langle 1, 0, \ldots, 0 \,;\, n \rangle &= \mathrm{A}_m(n) \\ \langle 0, 0, \ldots, 0 \,;\, n \rangle &= n \\ \langle a_m, a_{m-1}, \ldots, a_0 + 1 \,;\, n \rangle &= \langle a_m, a_{m-1}, \ldots, a_0 \,;\, 2n \rangle \\ \langle a_m, a_{m-1}, \ldots, a_{k+1} + 1, \underbrace{0, \ldots, 0}_{k+1 \text{ zeros}} \,;\, n \rangle &= \langle a_m, a_{m-1}, \ldots, a_{k+1}, n, \underbrace{0, \ldots, 0}_{k \text{ zeros}} \,;\, 1 \rangle. \end{aligned} \tag{1.1}$$

These equalities, read from left to right, can be naturally transformed into *computation rules*. The first equality says that the computation starts with all normal counter equal 0 except the left-most one which equals 1. The second equality says that the computation ends when all normal counters have value 0; the special counter shows then the result. The third equality

(cf. the definition of A_0) describes the computation rule that allows to decrease by 1 the right-most normal counter and to double the special one. Finally, the last equality (cf. the recursive definition of A_{k+1}) describes a computation rule that is enabled if the right-most normal counter has value 0; it allows to decrement the right-most non-zero normal counter by 1, increment the next (necessarily empty) counter by n, where n is the current value of the special counter, and set the special counter to 1. Note that it never happens that two or more different rules are enabled simultaneously.

Clearly, the computation rules can be implemented on a counter machine. But what results can we obtain, if the machine is lossy? The following fact implies that the results so obtained are smaller or equal to the correct one (by $\sqsubseteq$ we denote the pointwise order on tuples of integers of the same length):

FACT 1. If $(a_m, \ldots, a_0) \sqsubseteq (a'_m, \ldots, a'_0)$ and $n \leq n'$ then $\langle a_m, \ldots, a_0 \,;\, n \rangle \leq \langle a'_m, \ldots, a'_0 \,;\, n' \rangle$.

Indeed, the computation of the value of $\langle a_m, a_{m-1}, \ldots, a_0 \,;\, n \rangle$ on a lossy machine can be organised so that every loss corresponds to a decrease of some of arguments $a_m, \ldots, a_0$. Therefore a lossy counter machine using $m + \mathcal{O}(1)$ counters may *weakly* compute A_m, i.e, given an input n compute, as a result, the correct value $A_m(n)$ or any smaller value. The machine uses, in addition to $m + 2$ above-mentioned counters, an additional counter in order to implement the third rule which doubles the special counter, and the last rule, which essentially moves the value of the special counter to a normal counter.

More importantly, we observe that the equalities (1.1) can be also read from right to left, again inducing (reverse) computation rules. The first equality says that the computation ends with the left-most counter equal to 1, all normal counters equal to 0, and the special counter equal to n (this value n will be specified below). The second equality says that the computation starts with all normal counters equal to 0. The third equality describes the computation rule that is enabled if the special counter is even, and allows to divide its value by 2 and to increment the right-most normal counter by 1. Finally, the last equality describes a computation rule that is enabled if the special counter is equal 1, and allows to move the value of the right-most non-zero normal counter ($a_k = n$) to the special counter, set the counter a_k to 0, and increment the next counter to the left (a_{k+1}) by 1. Note that it may never happen that both the third and the last rule are enabled simultaneously.

As before, the reverse computation rules can be implemented on a counter machine. Intuitively speaking, a lossy counter machine using $m + \mathcal{O}(1)$ counters can weakly compute inverse of A_m. As before, the machine uses an additional counter to implement a subroutine that tests if the last rule is enabled—i.e., the special counter is 1—or the third rule is enabled—i.e., the special counter is positive and even (cf. Problem 2).

A run of a counter machine is *bounded by* ℓ if the sum of values of all the counters is at most ℓ along the run. We are now prepared for proving the lower bound of Theorem 2, by reduction from the following decision problem:

Input:	A deterministic counter machine $\mathcal{M}$ of size n.
Question:	Is the only run of $\mathcal{M}$ halting and bounded by $A(n)$?

The problem is complete in the class ACK of problems
solvable in time (or space, there is no difference) equal to

Ackermann function applied to any primitive recursive function applied to the input size[8] (that is $A(p(n))$, for an arbitrary primitive recursive function p) under primitive recursive reductions (that is, reductions computable in primitive recursive time, or space).

[8] Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016

Given a counter machine $\mathcal{M}$ of size n, the reduction yields a lossy counter machine $\widetilde{\mathcal{M}}$ consisting of three parts $\mathcal{M}_A$, $\mathcal{M}'$ and $\mathcal{M}_A^{rev}$ which are composed sequentially. The first and the last parts weakly compute A_n and its reverse, respectively, as discussed above, and use $n + \mathcal{O}(1)$ counters each. The middle part $\mathcal{M}'$ simulates the machine $\mathcal{M}$ while running, intuitively speaking, *on a budget*, as described below. The machine $\mathcal{M}'$ uses an additional counter b whose initial value constitutes a budget, i.e., a bound on the sum of all counters along a run. At every decrement of an arbitrary counter in $\mathcal{M}$, the machine $\mathcal{M}'$ increments the counter b; symmetrically, at every increment of an arbitrary counter in $\mathcal{M}$, the machine $\mathcal{M}'$ decrements the counter b (or, if $b = 0$, it enters an additional *error state*). Note that $\mathcal{M}'$ is lossy and hence, intuitively speaking, its budget may also decrease spontaneously during a run. W.l.o.g. we may assume that whenever M enters its halting state, all its counters equal 0. Due to this assumption, if the counter b is not decreased spontaneously due to loss, its final value is the same as the initial one.

The three parts of $\widetilde{\mathcal{M}}$ are composed as follows. The special counter of $\mathcal{M}_A$ is identified with the budget counter b of $\mathcal{M}'$, and also with the special counter of $\mathcal{M}_A^{rev}$. Machine $\widetilde{\mathcal{M}}$ starts by incrementing the left-most normal counter of $\mathcal{M}_A$ by 1, and incrementing the special counter of $\mathcal{M}_A$ by n (thus reaching a configuration corresponding to $\langle 1, 0, \dots, 0 ; n \rangle$, albeit possibly decreased due to losses) and then it runs $\mathcal{M}_A$ (which thus weakly computes $A_n(n) = A(n)$). This part ends once all normal counter have value 0 (thus reaching a configuration $\langle 0, 0, \dots, 0 ; \ell \rangle$ with $\ell \leq A(n)$).

Once $\mathcal{M}_A$ ends, the control is transferred to the machine $\mathcal{M}'$ that performs its computation until a halting or error state is reached. In the former case the control is transferred to the machine $\mathcal{M}_A^{rev}$ (hence it starts in a configuration corresponding to $\langle 0, 0, \dots, 0 ; \ell \rangle$ for some $\ell \leq A(n)$). The machine performs its computation until its left-most normal counter gets value 1 (the intention is to reach a configuration corresponding to $\langle 1, 0, \dots, 0 ; n \rangle$) or no reverse computation rule is enabled (in which case $\mathcal{M}_A$ enters an error state). In the first case the machine checks if the special counter equals n, and moves to the halting state.

A crucial observation is that every halting run of $\widetilde{\mathcal{M}}$ is perfect, i.e., without losses. Indeed, any loss in any of the three parts is non-revertible and prevents $\widetilde{\mathcal{M}}$ from reaching the final halting state. This observation underlies correctness of our reduction:

LEMMA 4. The following conditions are equivalent:

1. The only run of $\mathcal{M}$ is halting and bounded by $A(n)$.
2. $\widetilde{\mathcal{M}}$ has a halting run.

1.3 *Problems*

PROBLEM 1. Show that a quasi order is a WQO if, and only if, it has no infinite descending chains (i.e., it is well-founded) and no infinite antichains.

Problem 2. Design the details of lossy counter machines that weakly compute Ackermann function and its inverse.

Problem 3. Design a procedure to check if a given antichain in $Q \times \mathbb{N}^d$ is a non-halting witness, as defined in Section 1.1.

Problem 4. Do the results of Chapter 1 hold for *nondeterministic* lossy counter machines?

Problem 5. Do the results of Chapter 1 hold for *gainy* counter machines which witness spontaneous increments of counters instead of spontaneous decrements?

Problem 6. Show that a Wqo $(X, \preceq)$ has no infinite strictly increasing sequence of upward closed subsets, i.e., there is not infinite sequence

$$U_1 \subset U_2 \subset U_3 \subseteq \dots$$

where each $U_i \subseteq X$ is upward closed with respect to $\preceq$. Is this condition sufficient for $(X, \preceq)$ to be a Wqo?

Problem 7. Desing a procedure that computes the set of configurations reverse reachable from a given configuration in a lossy counter machine, and use it to obtain a decision procedure for the halting problem.

Problem 8. Show decidability of the following *non-termination* problem:

Input:	A nondeterministic lossy counter machine $\mathcal{M}$.
Question:	Does $\mathcal{M}$ have an infinite run?

Problem 9. Show undecidability of the following repeated (Büchi) reachability problem:

Input:	A nondeterministic lossy counter machine $\mathcal{M}$ and a control state $q \in Q$.
Question:	Does $\mathcal{M}$ have an infinite run which visits q infinitely often?

Note close similarity of repeated reachability and non-termination from the previous problem. *Hint*: repetitive simulation of a counter machine on an unbounded initial budget.

Problem 10. Prove that the set Σ^* of all words over a finite alphabet Σ, ordered by subsequence ordering, is a Wqo.

Problem 11. Show that every language closed under removing letters from its words is regular.

Problem 12. A lossy *FIFO* automaton consists of a finite number of finite-state machines that communicate (asynchronously) through a finite number of FIFO buffers, which witness spontaneous disappearances of letters. Show that emptiness tests of a FIFO automaton may be eliminated.

Problem 13. Reduce the halting problem of a multi-party lossy FIFO automaton to a single-party lossy automaton with a single FIFO.

Problem 14. Reprove the results of Chapter 1 for lossy FIFO automata. What about lossy FIFO VASS?

PROBLEM 15. Adapt the decidability argument of Chapter 1 to the coverabiliy problem in VAS. Try formulate abstract conditions sufficient for correctness of the negative semi-decision procedure.

Bibliography

[1] Richard Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, 2003.

[2] Marvin Minsky. Recursive unsolvability of Post's problem of 'tag' and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961.

[3] Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.

[4] Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *Proc. MFCS 2010*, volume 6281 of *LNCS*, pages 616–628. Springer, 2010.