Lossy counter machines and timed automata

Sławomir Lasota

Course notes

January 8, 2025

Contents

1 *Lossy counter machines* 5

1.1 Decidability of the halting problem 7

1.2 Non-primitive recursive lower bound 9

1.3 Problems 11

2 *Timed automata* 15

2.1 Decidability of emptiness 17

2.2 Undecidability of universality 18

2.3 Problems 21

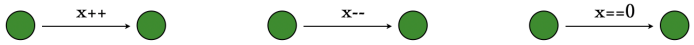
Bibliography 23

1
Lossy counter machines

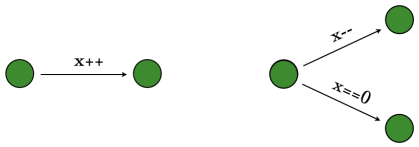
We start by recalling the model of deterministic *counter machines* (called also *Minsky machines*[1]).

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.

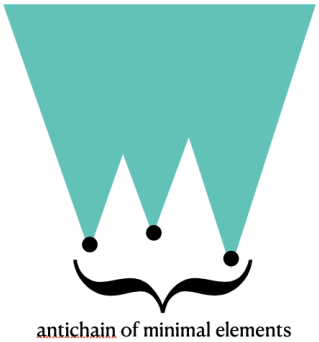
Let Q be the set of control states of a lossy counter machine. We deduce that the extension $\sqsubseteq$ of the pointwise ordering of vectors to the configurations $Q \times \mathbb{N}^d$, where d is the number of counters, is a WQO too:

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:

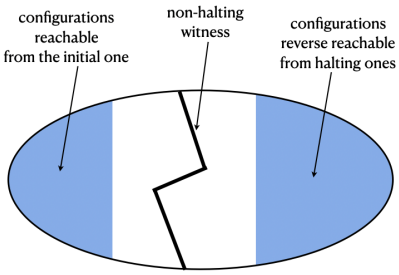
Lemma 3. The machine does not halt if, and only if, the machine admits a finite non-halting witness as defined above.

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.

2
Timed automata

In this chapter we present an extension[1] of finite automata that takes into account time elapsing between the input letters.

In the presented approach, the automaton's input consists of pairs of the form $(a, t) \in \Sigma \times \mathbb{R}_+$, where $a \in \Sigma$ is a letter from a finite alphabet, and $t \in \mathbb{R}_+ = \{r \in \mathbb{R} \mid r \geq 0\}$ is a nonnegative real *timestamp*. Finite (or infinite) sequences

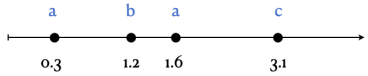
$$(a_1, t_1), (a_2, t_2), \ldots, (a_n, t_n)$$

of such pairs, satisfying the monotonicity condition: $t_1 \leq t_2 \leq \ldots \leq t_n$, we call *timed words* over an alphabet Σ, and sets of timed words we call *timed languages*.

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

Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994

An example timed word:



How the amount of time elapsed between letters can be measured? To this aim we will use real-values variables called *clocks*. A clock can be reset to 0 at any moment, and then its value increases with the elapse of time until a next reset. Thus at every moment the clock value (age) shows the amount of time elapsed since its last reset. The clock value can be tested using (in)equalities between clocks and integer constants, for instance

$$x > 1 \qquad x > y + 2 \qquad x \geq y - 1 \qquad x = 3.$$

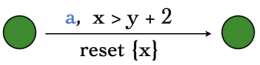
Formally, a *clock constraint* is any Boolean combinations of inequalities of the form

$$x > n \qquad x > y + n$$

where x, y are clocks and $n \in \mathbb{Z}$ is an integer. We use syntactic sugar and write, e.g., $x = y + 3$ instead of $\neg(x > y + 3 \vee x < y + 3)$.

A *timed automaton* is like an ordinary finite automaton, but additionally equipped with a finite set of *clocks* $X = \{x, y, z, \ldots\}$. Moreover, each transition of a timed automaton is equipped with a clock constraint (which must be satisfied in order to fire a transition) and a subset of clocks to be reset.

An example transition:



The input letter and the clock constraint are above the arrow, and the reset clocks below.

Formally, a transition is a quintuple $\langle q, a, \varphi, Y, q' \rangle$ where q, q' are control states of the timed automaton, $a \in \Sigma$ is an input letter, φ is a clock constraint, and $Y \subseteq X$ is an arbitrary subset of clocks.

For defining semantics of timed automata we need to define *configurations* $\langle q, v \rangle$, which consist is a control state q and a clock valuation $v : X \to \mathbb{R}_+$. Intuitively, $v(x)$ is the age of the clock x, i.e., the amount of time elapsed since its last reset. A transition $\langle q, a, \varphi, Y, q' \rangle$ induces an a-step from a configuration $\langle q, v \rangle$ to a configuration $\langle q', v' \rangle$ if $v \models \varphi$ and $v' = v[Y \mapsto 0]$ is obtained from v by setting $v(x) = 0$ for all $x \in Y$. Furthermore, for any $t \in \mathbb{R}_+$ we have also a t-time elapse step from $\langle q, v \rangle$ to $\langle q, v' \rangle$ where $v' = v + t$ is obtained from v by adding t to all clocks: $v'(x) = v(x) + t$ for all $x \in X$. Clearly, time elapse steps do not change control state. The steps we denote, respectively, as:

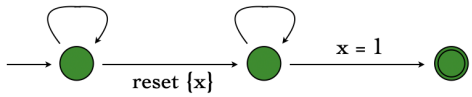
$$\langle q, v \rangle \xrightarrow{a} \langle q', v' \rangle \qquad\qquad \langle q, v \rangle \overset{t}{\rightsquigarrow} \langle q, v' \rangle.$$

A timed word $(a_1, t_1)(a_2, t_2) \dots (a_n, t_n)$ is accepted by a timed automaton if it has a sequence of alternating a_i-steps and $(t_{i+1} - t_i)$-time elapse steps of the following form (put $\delta_i = t_i - t_{i-1}$):

$$\langle q_0, v_0 \rangle \overset{t_1}{\rightsquigarrow} \langle q_0, v_0 + t_1 \rangle \xrightarrow{a_1} \langle q_1, v_1 \rangle \overset{\delta_2}{\rightsquigarrow} \langle q_1, v_1 + \delta_2 \rangle \xrightarrow{a_2} \langle q_2, v_2 \rangle \quad \dots \quad \overset{\delta_n}{\rightsquigarrow} \langle q_{n-1}, v_{n-1} + \delta_n \rangle \xrightarrow{a_n} \langle q_n, v_n \rangle,$$

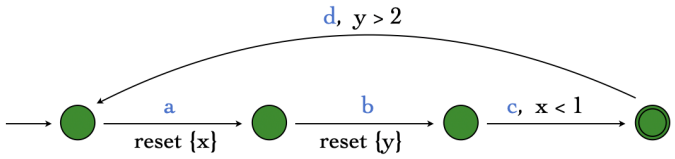
for some control states $q_0, q_1, \dots, q_n$ where q_0 is an initial state and q_n is an accepting one, and some clock valuations $v_0, v_1, \dots, v_n$. We assume that $v_0(x) = 0$ for all $x \in X$ thus, intuitively speaking, a timed automaton starts will ages of all clocks equal 0. Analogously one may define the language of timed ω-words recognized by a timed automaton.

Example 1. What is the language recognized by the following timed automaton with one clock? The alphabet is a singleton and hence we omit input letters. We omit also trivial clock constraints $\varphi =$ true vacuously true, and trivial reset sets $Y = \emptyset$.



The automaton accepts words that contain two timestamps of difference equal to 1.

Example 2. What is the language recognized by the following timed automaton with two clock? The alphabet is $\Sigma = \{a, b, c, d\}$.



Remark 3. Why we only allow integer constants in clock constraints, and not rational or even real ones? First, as we want timed automata to be finitely representable, we should restrict to rational constants. Furthermore, timed automata with rational constants are essentially equally expressive as ones with integer constants, see Problem 16.

2.1 *Decidability of emptiness*

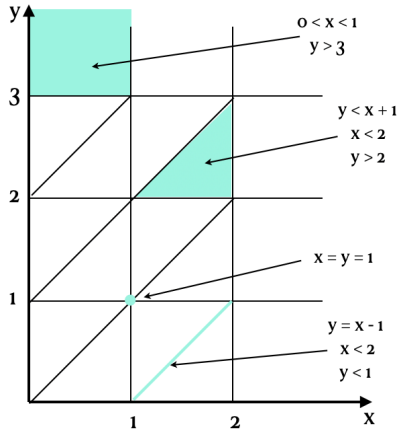
The configuration space of a timed automaton is infinite (uncountable in fact). Nevertheless, some decision problems, including the most central nonemptiness problem, are decidable due to a construction called *regions*.

THEOREM 3. The nonemptiness problem is decidable for timed automata.

For proving the theorem, consider a fixed timed automaton $\mathcal{A}$ with clocks X. Without loosing generality assume that the automaton uses no *diagonal* clock constraints, i.e., inequalities of the form $x > y + n$ (see Problem 19). As disjunction may be eliminated by distributing conjunct to separate transitions, we may assume that every constraint in $\mathcal{A}$ is a conjunction of constraints of the form $x > n$ or $x = n$ or $x < n$.

For each clock x let n_x be the greatest value of a constant to which x is compared in a constraint. We call n_x the threshold of x. We consider two clock valuations $v, u : X \to \mathbb{R}_+$ as equivalent if, intuitively speaking, (i) the same clocks have values below their thresholds in v and u, (ii) the integer parts of clock values are the same in v and u, and (iii) the fractional parts of clocks are ordered in the same way in v and u. Furthermore, the last two conditions only apply to clocks with value below the threshold.

Clock regions for two clocks x, y and $n_x = 2$ and $n_y = 3$:



We write below $\lfloor r \rfloor$ and $\langle r \rangle$ for integer and fractional part of $r \in \mathbb{R}$, respectively. Formally, we define an equivalence over clock valuations as follows: $v \equiv u$ if

(i) $\forall x \in X \,.\, v(x) \leq n_x \iff u(x) \leq n_x$

(ii) $\forall x \in X \,.\, v(x) \leq n_x \implies \lfloor v(x) \rfloor = \lfloor u(x) \rfloor \wedge \big(\langle v(x) \rangle = 0 \iff \langle u(x) \rangle = 0\big)$

(iii) $\forall x, y \in X \,.\, v(x) \leq n_x \wedge v(y) \leq n_y \implies \big(\langle v(x) \rangle < \langle v(y) \rangle \iff \langle u(x) \rangle < \langle u(y) \rangle\big)$.

Equivalence classes of $\equiv$ we call *clock regions*. The number thereof is finite, but exponential with respect to the number of clocks. Some of regions are bounded, and some are not. Observe that every bounded region is a minimal nonempty set of clock valuations definable using (possibly diagonal) clock constraints, i.e., a bounded clock region can't be further split into smaller subsets definable using such constraints.

FACT 2. if $v \equiv v'$ then:[2]

- for every clock constraint φ in $\mathcal{A}$, $v \models \varphi \iff v' \models \varphi$,
- for every $Y \subseteq X$, $v[Y \mapsto 0] \equiv v'[Y \mapsto 0]$,
- for every $t \in \mathbb{R}_+$ there is some $t' \in \mathbb{R}_+$ such that $v + t \equiv v' + t'$.

[2] **Question:** How to adapt the definition of regions so that the fact holds for $\mathcal{A}$ having diagonal constraints?

Relying on the fact we build the region automaton $R(\mathcal{A})$, a nondeterministic finite automaton over Σ. Its states are *regions*, i.e., pairs (q, R) where q is a control state and R a clock region.

Relying on the first point in the above fact we may write $R \models \varphi$ for a clock region R, and relying on the second point we may speak of a clock region $R' = R[Y \mapsto 0]$ obtained from R by reseting a subset $Y \subseteq X$ of clocks. We define transitions of $R(\mathcal{A})$ as all triples of the form:

$$(q, R) \xrightarrow{a} (q', R')$$

such that $\mathcal{A}$ has a transition $\langle q, a, \varphi, Y, q' \rangle$ with $R \models \varphi$ and $R' = R[Y \mapsto 0]$. Furthermore, due to the last point in the above fact we may write $R \rightsquigarrow R'$ for clock regions R, R' if for some $v \in R$ and $t \in \mathbb{R}_+$ we have $v + t \in R'$. In order to account for time elapse in $\mathcal{A}$, the region automaton $R(\mathcal{A})$ has ε-transitions of the form

$$(q, R) \xrightarrow{\varepsilon} (q, R'),$$

for every control state q, and regions R, R' satisfying $R \rightsquigarrow R'$. Initial states of $R(\mathcal{A})$ are those pairs (q, R) where q is initial in $\mathcal{A}$ and R contains only one clock valuation constantly equal to 0. Accepting states of $R(\mathcal{A})$ are those pairs (q, R) where q is accepting in $\mathcal{A}$. The nonemptiness of $\mathcal{A}$ reduces to the nonemptiness of $R(\mathcal{A})$:

Lemma 5. The language of $\mathcal{A}$ is nonempty iff the language of $R(\mathcal{A})$ is nonempty.

The *only if* direction is immediate, while the *if* direction relies on the last point in Fact 2. As the construction of $R(\mathcal{A})$ is effective, we thus complete the proof of Theorem 3.

Remark 4. The non-emptiness problem is PSpace-complete (see Problem 21). The original hardness proof[3] requires 3 clocks, while a recent beautiful construction of Fearnley and Jurdziński[4] requires just 2 clocks.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994

[4] John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015

2.2 *Undecidability of universality*

Languages of timed automata are not closed under complement, and therefore timed automata do not determinize (see Problem 22). We now inspect a closely related issue - undecidability of the universality problem (does a given timed automaton accept all timed words over its input alphabet?).

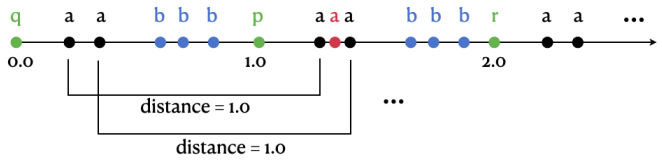
Theorem 4. The universality problem is undecidable for timed automata.

In consequence of this result we easily deduce undecidability of language containment and language equality problems.

We prove Theorem 4 by reduction from the halting problem of 2-counter Minsky machines (cf. the beginning of Chapter 1). Given such a machine $\mathcal{M}$, we are going to construct a timed automaton $\mathcal{A}$ with two clocks which is not universal if, and only if, $\mathcal{M}$ halts. The negation is necessary as both the halting problem and the non-universality problem are semi-decidable.

Let Q denote states of $\mathcal{M}$. For simplicity we assume, without loosing generality, that $\mathcal{M}$ has no two different transitions with the same source and target control states. Before proving a construction we need to define the notion of an *encoding* of a run of $\mathcal{M}$ in a timed word over the alphabet $\Sigma = Q \cup \{a, b\}$. In the encoding, consecutive half-open unit intervals $[0, 1), [1, 2), \ldots$

store a description of consecutive configurations in a run of $\mathcal{M}$. A configuration (q, n, m) of $\mathcal{M}$, consisting of a state $q \in Q$ and values n, m of the two counters, is represented by q with integer timestamp (left end of an interval) followed by n letters a, and then by m letters b, with strictly increasing timestamps arbitrarily chosen inside the unit interval.



In the example shown in the picture, the first configuration is $(q, 2, 3)$ and the second one is $(p, 3, 3)$. Thus the following *structural consistency* holds: letters from Q have consecutive integer timestamps, and other letters from $\{a, b\}$ have non-integer ones, and untiming of a word belongs to the regular language $(Q\, a^*\, b^*)^*$, i.e., letters a preced letters b within each unit interval.

Furthermore, on every pair of consecutive unit intervals we impose the following *semantic consistency*, (corresponding, intuitively speaking, to the successor relation on configurations of a Minsky machine). Consider a fixed pair consecutive unit intervals, and let q and p be the control states in the first and in the second interval, respectively. First of all, letters a appearing in the first interval can be matched, one-to-one, to letters a in the second interval, in such a way that timestamps of matched letters differ by exactly 1, and likewise for letters b, except for at most one letter a or b in some of the two intervals that may be unmatched (red a is unmatched in the picture above). Furthermore, only last a or last b may be unmatched (the red a in the picture is not the last one, so the semantics consistency is violated by this timed word). Finally, the last a (resp. b) in the first interval is unmatched only if the corresponding transition of $\mathcal{M}$ from q to p increments the first (resp. the second) counter. Likewise, the last a (resp. b) in the second interval is unmatched only if the corresponding transition of $\mathcal{M}$ decrements the first (resp. the second) counter. Intuitively, full match represents a step of $\mathcal{M}$ which preserves values of both counters, while an unmatched letter a or b corresponds to decrement or increment of one of counters.

We observe that a timed word satisfying the two consistency conditions *encodes* a run ρ of $\mathcal{M}$ (such a timed word we call an encoding of a run ρ); reciprocically, every run of $\mathcal{M}$ has an encoding (actually infinitely many different encodings). Let $\textsc{Runs}_{\mathcal{M}}$ be the set of all timed words that are encodings of accepting runs of $\mathcal{M}$. Our reduction computes, given $\mathcal{M}$, a timed automaton $\mathcal{A}_{\mathcal{M}}$ that recognizes the complement of $\textsc{Runs}_{\mathcal{M}}$.

LEMMA 6. There is a timed automaton $\mathcal{A}_{\mathcal{M}}$ recognizing the complement of $\textsc{Runs}_{\mathcal{M}}$.

Proof sketch. The automaton $\mathcal{A}_{\mathcal{M}}$ uses nondeterminism to choose which of the conditions is violated: either the first control state is not initial, or the last one is not accepting, or any of the two consistency conditions fail. The structural consistency (or its violation) is checkable by a deterministic timed automaton with 1 clock. For the semantic consistency, the automaton $\mathcal{A}_{\mathcal{M}}$ additionally uses nondeterminism to guess a pair of consecutive unit intervals and a position therein where the condition is violated (the pair of intervals and a position need not be unique).

Let $q, p \in Q$ denote the control states in the two intervals and let t be the (unique) transition of $\mathcal{M}$ from q to p. We distinguish several cases of violation.

1. The first case of violation is an unmatched not-last a in the first interval (on the left in the picture below); or an unmatched last a (on the right in the picture below) while t is not decrementing the first counter:

   Likewise for unmatched b. This condition is checkable using 1 clock.

2. The second case is an unmatched letter in the second interval, say a, which is a non-last one (on the left in the picture below); or an unmatched last one (on the right in the picture below) while t is not incrementing the respective counter:

   Likewise for unmatched b. This condition is checkable using 2 clocks (how?).

3. The third case is when the last a in the first interval is matched (or there is no a in the first interval), while t is decrementing the first counter, and likewise for the second counter. This is checkable with 1 clock.

4. The fourth case is when the last a in the second interval is matched (or there is no a in the second interval), while t is incrementing the first counter, and likewise for the second counter. This is checkable with 1 clock.

5. The last fifth case is when t is zero-testing the first counter while the first interval contains some a, and likewise for the second counter. This is checkable with no clock.

We observe that a timed word satisfying the structural consistency violates the semantic one if, and only if, some pair of consecutive unit intervals and some position therein satisfies one of the five conditions listed above. □

Remark 5. The undecidability proof shown in this section works for timed automata with 2 clocks. One can use a variant of the region construction for a symbolic determinization of timed automata with 1 clock, which allows us to regain decidability[5] of the universality problem. Likewise one proves decidability of language containment and equality for timed automata with 1 clock. Concerning the lower bound, one can show non-primitive recursive lower bound for automata with 1 clock (see

[5] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proc. (LICS 2004*, pages 54–63, 2004

Problem 26) by adaptation of the undecidability proof of this section.

2.3 Problems

PROBLEM 16. Are timed automata with rational constants essentially more expressive? Is the nonemptiness problem for those generalized timed automata reducible to the nonemptiness problem for 'ordinary' timed automata?

PROBLEM 17. Do ε-transitions increase the expressive power of timed automata?

PROBLEM 18. Show that every timed automaton is equivalent to a one without diagonal constraints. What is the blowup of automaton size?

PROBLEM 19. Show that time abstraction of the language L of a timed automaton, defined as $\{a_1a_2\dots a_n \in \Sigma^* \mid \exists t_1, t_2, \dots, t_n \in \mathbb{R}_+ \,.\, (a_1, t_1)(a_2, t_2)\dots(a_n, t_n) \in L\}$, is a regular language.

PROBLEM 20. Show decidability of the non-emptiness problem[6] for timed automata with two clocks x, y, extended with *additive* constraints of the form $x + y \leq n$ where $n \in \mathbb{N}$.

[6] Interestingly, the problem becomes undecidable for 4 clocks, while its status is unknown for 3 clocks.

PROBLEM 21. Prove that the nonemptiness problem for timed automata is PSPACE-complete, NP-hard[7] for automata with 2 clocks and solvable in polynomial time for automata with 1 clock.

[7] It is actually PSPACE-complete even for 2 clocks, as shown in:

John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015

PROBLEM 22. Find a non-complementable timed automaton, i.e., recognising a language whose complement is not recognized by a timed automaton.

PROBLEM 23. Propose a definition of *deterministic* timed automata and use the automaton from Problem 22 to show that deterministic timed automata are less expressive than nondeterministic ones.

PROBLEM 24. Using Higman's lemma, show decidability[8] of the universality problem of timed automata with one clock.

[8] The details concerning solutions of this and the next problem are to be found in:

Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proc. (LICS 2004*, pages 54–63, 2004

PROBLEM 25. Refine the construction from the previous problem to derive decidability of language containment $L(\mathcal{A}) \subseteq L(\mathcal{B})$ for a timed automaton $\mathcal{A}$ and a timed automaton $\mathcal{B}$ with one clock.

PROBLEM 26. Adapt the reduction from Section 2.2 to prove non-primitive recursive complexity[9] of the universality problem for timed automata with 1 clock. Use the lower bound of Section 1.2.

[9] The details concerning solutions of this and the next problem are to be found in:

Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2):10:1–10:27, 2008

PROBLEM 27. Adapt further the reduction to show undecidability of the universality problem for Büchi timed automata with 1 clock.

Bibliography

[1] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.

[2] John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015.

[3] Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Log.*, 9(2):10:1–10:27, 2008.

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

[5] 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.

[6] Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proc. (LICS 2004*, pages 54–63, 2004.

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

[8] 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.