On Parametric Timed Automata and One-Counter Machines

Daniel Bundala[a,*], Joel Ouaknine[a]

[a] *Department of Computer Science, University of Oxford, United Kingdom*

Abstract

Two decades ago, Alur, Henzinger, and Vardi introduced the reachability problem for parametric timed automata in the seminal paper [2]. Their main results are that reachability is decidable for timed automata with a single parametric clock, and undecidable for timed automata with three or more parametric clocks.

In the case of *two* parametric clocks, decidability was left open, with hardly any progress (partial or otherwise) that we are aware of in the intervening period. As pointed out by Alur *et al.*, this case also subsumes Ibarra *et al.*'s reachability problem for "simple programs" [14], another long-standing open problem, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

In this manuscript we establish a correspondence between reachability in parametric timed automata with at most two parametric clocks (and arbitrarily many nonparametric clocks) and reachability for a certain class of parametric one-counter machines. We then leverage this connection (i) to improve Alur *et al.*'s decision procedure for one parametric clock from nonelementary to 2NEXP; (ii) to show decidability for two parametric clocks provided the timed automaton uses only a single parameter; (iiii) to show decidability for various resulting classes of parametric one-counter machines; and (iv) to show decidability of reachability for the simple programs of Ibarra *et al.* in the presence of a single parameter. In addition, we prove that for one and two parametric clocks the reachability problem is NEXP-hard and $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hard respectively.

Keywords: Parametric Timed Automata, One-Counter Machines, Reachability

1. Introduction

The problem of reachability in parametric timed automata was introduced over two decades ago in a seminal paper of Alur, Henzinger, and Vardi [2]: given a timed automaton in which some of the constants appearing within guards on transitions are parameters, is there some assignment of integers to the parameters such that an accepting location of the resulting concrete timed automaton becomes reachable?

As is well known, embedded computer systems often depend on the precise timing of individual actions. A standard formalism for modeling such time-dependent systems is that of timed automata—a class of finite automata extended with clocks. In a timed automaton, all clocks evolve simultaneously, at the same rate, and each clock can be reset individually by edges of the automaton. Such edges also usually incorporate timed guards, i.e., simple clock expressions which must be satisfied in order for the corresponding transitions to be enabled, thus constraining the possible evolutions of the timed system.

Getting such timing constraints right is however difficult. Embedded systems further operate in a surrounding environment, and therefore leaving some constraints under-specified in the timed-automata models allows us to elicit the environments in which the system behaves as required. Such considerations led to the introduction of *parametric timed automata* [2] in which some of the constraints feature unspecified parameters. The goal is then to find values of the parameters for which the timed automaton behaves as intended.

*Corresponding author; address: Department of Computer Science, University of Oxford, Oxford, United Kingdom
Email addresses: bundala@gmail.com (Daniel Bundala), joel@cs.ox.ac.uk (Joel Ouaknine)

As one might expect, verification of safety properties leads to the reachability problem for parametric timed automata: is a bad state reachable for some values of the parameters? This problem, which is the focus of this manuscript, was first considered by Alur, Henzinger and Vardi [2] over two decades ago, and shown to be undecidable in full generality. More precisely, given a parametric timed automaton, a clock is said to be *nonparametric* if it is never compared with a parameter, and is *parametric* otherwise. Alur *et al.* showed that, for timed automata with a single parametric clock, reachability is decidable (irrespective of the number of nonparametric clocks). However, the decision procedure given in [2] has provably nonelementary complexity. In addition, Alur *et al.* showed that reachability becomes undecidable for timed automata with at least three parametric clocks.

The decidability of reachability for parametric timed automata with *two* parametric clocks (and arbitrarily many nonparametric clocks) was left open in [2], with hardly any progress (partial or otherwise) that we are aware of in the intervening period. Alur *et al.* further showed that this problem subsumes the question of reachability in Ibarra *et al.*'s "simple programs" [14], also open for over 20 years, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

In this paper we substantially improve the decision procedure for one parametric clock (from nonelementary to 2NEXP) and give decidability results for classes of two parametric clocks. In this process—and as a main contribution of interest in its own right—we establish a correspondence between reachability in parametric timed automata with at most two parametric clocks and reachability for a certain class of parametric one-counter machines, a formalism which lends itself more conveniently to analysis. In particular, we use this connection to substantially improve Alur *et al.*'s decision procedure, identify the first nontrivial decidable fragment with two parametric clocks, and make progress on reachability for the simple programs of Ibarra *et al.*

It is worth noting that similar connections have previously appeared in nonparametric settings [11], and subsequently used to determine the precise complexity of reachability in two-clock (nonparametric) timed automata [7]—another long-standing open problem.

None of our results impose any restriction on the number of nonparametric clocks. More precisely, we show that:

1. The reachability problem for parametric timed automata with a single parametric clock is equivalent to the reachability problem for parametric one-counter machines with updates $-1, 0$ and 1, and comparisons $\leq p_i, \geq p_i$ for parameters p_i.

2. The reachability problem for parametric timed automata with two parametric clocks is equivalent to the reachability problem for parametric one-counter machines with updates $\pm c_i, \pm p_i$, comparisons $\leq p_i, \geq p_i$ and a few other technical operations for constants $c_i \in \mathbb{N}$ and parameters p_i.

In Sec. 7, we then use the relationship between parametric timed automata and the respective classes of parametric one-counter machines to show that:

3. In the case of a single parametric clock (with arbitrarily many nonparametric clocks and arbitrarily many parameters), the reachability problem is in 2NEXP and NEXP-hard improving the nonelementary decision procedure of Alur *et al.*

4. The reachability problem is decidable for the class of parametric one-counter machines mentioned in (1).

5. The reachability problem is decidable for parametric timed automata with two parametric clocks (and arbitrarily many nonparametric clocks), if the automaton uses only a *single* parameter. Further, the problem is $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hard.

6. We show that the reachability problem is decidable for the class of parametric one-counter machines mentioned in (2) if they use only a single parameter.

7. We use the techniques developed to solve the reachability problem for the simple programs of Ibarra *et al.* provided they use only a single parameter.

At the time of writing, the result in item 6 corresponds do the largest decidable fragment of parametric timed automata with two parametric clocks. Our decidability results build upon new developments in the theory of one-counter machines [10] and their encodings in Presburger arithmetic [9].

As in [2], our results are presented for timed automata interpreted over discrete time. However, for parametric timed automata with *closed* (i.e., *non-strict*) clock constraints and parameters restricted to ranging over integers,[1] standard digitisation techniques apply [12, 23], reducing the reachability problem over dense time to discrete (integer) time.

2. Related Work

The decidability of reachability for parametric timed automata has been previously achieved in certain restricted settings. The primary concern in such restricted settings was usually the development of practical verification tools, and indeed the resulting algorithms tend to have comparatively good complexity. However, solutions to these special cases do not lead to general decision problems as is the case in this manuscript. For instance, decidability can be achieved by bounding the allowed range of the parameters [15] or by requiring that parameters only ever appear either as upper or lower bounds, but never as both [13]: in the latter case, if there is a solution at all then there is one in which parameters are set either to zero or infinity.

Miller [21] observed that over dense time and with parameters allowed to range over rational numbers, reachability for parametric timed automata becomes undecidable already with a single parametric clock. In the same setting, Doyen [6] showed undecidability of reachability for two parametric clocks even when using exclusively open (i.e., strict) time constraints.

A connection between timed automata and counter machines was previously established in nonparametric settings [11], and exploited to show that reachability for (ordinary) two-clock timed automata is polynomial-time equivalent to the emptiness problem for one-counter machines, even when constants are encoded in binary. Unfortunately, it is not obvious how to extend and generalise this construction to parametric timed automata, specifically in the case of two parametric clocks and an arbitrary number of nonparametric clocks, as we handle in the present section. The reduction of [11] was used in [7] to show that emptiness for bounded one-counter machines, and hence reachability for two-clock timed automata, is PSPACE-complete, solving what had been a longstanding open problem.

It is known [21] that for one parametric clock but *no* nonparametric clocks the emptiness problem is NP-complete. Compare this to 2NEXP-completeness proved in this manuscript if nonparametric clocks are allowed. It is known that reachability in nonparametric timed automata is PSPACE-complete [3] and thus allowing nonparametric clocks automatically "extends the automaton with a PSPACE machine". We use this observation in our hardness results.

This manuscript is an extended version of [5] conference paper. Main technical contributions novel to the manuscript are the decidability results of item 6 and item 7 in the list above, to a proof of which is devoted Section 10. The decidability results obtained in this manuscript proceed via reduction to the existential fragment of Presburger arithmetic with divisibility. The satisfiability of such formulae has been long known to be decidable [19, 4]. In some previous works [10, 9] as well as in the conference paper [5] by the authors, it was claimed to be in NP. However, it was recently pointed out [18] that this is not correct; the misunderstanding coming mainly from the misunderstanding of an NP upper bound established [20]. The paper [20] is very technical, difficult to follow and, as pointed out in [18], the NP upped bound claimed in that paper applies only to formulas of a certain form and of *fixed* size. When applied to formulae of arbitrary size, the algorithm in [20] yields only 2NEXP decision procedure in the general case. It was recently showed [18] that the satisfiability is in NEXP and NP-hard. In particular, the NEXP upper bound claimed in [5] by the authors for the reachability in one-parametric-clock case is not correct. The decision procedure is correct, however, in the light of the recent findings the best provable upper bound is 2NEXP.

[1] Other researchers have considered variations in which parameters are allowed to range over rationals, yielding different outcomes as regards the decidability of reachability; see, e.g., [21, 6], discussed further below.

3. Preliminaries

We study and present several classes of automata in this manuscript. We begin by a general definition, which is then instantiated for various transition functions. In general, an *automaton* is a finite collection of *states* and *edges* (*transitions*) from one state to another. Formally,

Definition 3.1. *An* automaton A *is tuple* $A = (S, s_0, \Delta, F, Op, \lambda)$ *where*

- S *is the set of states,*
- s_0 *is the initial state,*
- $\Delta \subseteq S \times S$ *is the set of edges,*
- F *is the set of final states and*
- $\lambda : \Delta \to Op$ *is a function assigning a label from set* Op *to every edge.*

Whenever we define a class of automata in this manuscript, we specify only the set Op of allowed labels and their respective semantics.

3.1. Working with Paths

A *transition* $e = (q, q')$ of A is a tuple in the transition relation Δ where $q \in Q$ is the initial state of the transition, $\lambda(e) \in \Sigma$ is the letter associated with the transition and $q' \in Q$ is the final state of the transition. Two transitions (q_1, q_2) and (q_3, q_4) are *consecutive* if the final state of the first transition equals the initial state of the second transition: $q_2 = q_3$.

A *path* π through an automaton A is a sequence of transitions $(e_i)_{i=1}^{i=l}$ such that the transitions e_i and e_{i+1} are consecutive. The length of π, denoted $|\pi|$, equals l. The word $\lambda(e_1), \lambda(e_2), \ldots, \lambda(a_l)$ spelled by (associated with) π is denoted by $\mathrm{word}(\pi)$.

A path through finite automaton A is called *accepting* if it starts in the initial state q_0 and finishes in a final states $s \in F$. If π is nonempty path then we use $\mathrm{first}(\pi) = q_1$ to denote the *initial state* of π. We say that π starts (begins) in q_1. If π is finite and nonempty then we use $\mathrm{last}(\pi) = q'_l$ to denote the *final state* of π. We say that π finishes (terminates) in q'_l. If π is finite, we often write $\pi : q_1 \to q'_l$.

If the path π is written as $\pi = (e_i)_{i=1}^{i=l}$ and $\rho = (f_i)_{i=1}^{i=k}$ is a path such that the transitions e_l and f_1 are consecutive then the *concatenation* $\pi \cdot \rho$ of π and ρ is obtained by appending ρ after π. That is $\pi \cdot \rho = (g_i)_{i=1}^{i=l+k}$ where $g_i = e_i$ if $i \leq l$ and $g_i = f_{i-l}$ if $i > l$.

If $i \in \mathbb{N}$ is an index then $\pi(i)$ denotes the i-th state (starting from 1) of π. Precisely, $\pi(1) = q_1$ and $\pi(i) = q'_{i-1}$ for $i > 1$. For $a \leq b \in \mathbb{N}$ the expression $\pi[a \ldots b]$ denotes the subpath of π from index a to index b inclusively. The expression $\pi[a \ldots)$ denotes the suffix of π starting at index a.

A path π of length at least one is a *loop* if the first and the last state coincide: $\mathrm{first}(\pi) = \mathrm{last}(\pi)$ and $|\pi| > 0$. If π is a loop and $k \in \mathbb{N}$ then by π^k we denote the path obtained by concatenating π with itself k times: $\pi^k = \underbrace{\pi.\pi \ldots \pi}_{k \text{ times}}$.

4. Different Types of Automata

4.1. Timed Automata

A timed automaton is a finite automaton extended with a finite set of clocks C that all progress at the same rate and that can be individually reset to zero. Moreover, every transition is labelled by an expression of the form $\bigwedge_i c \bowtie d$ where $c \in C$ is a clock, $\bowtie \in \{\leq, =, \geq\}$ and $d \in \mathbb{N}$ is a constant. The set of all such expression over the set C of clocks is denoted $G(C)$.

Definition 4.1. *A* timed automaton A *over the set of clocks* C *is an automaton with*

$$Op = 2^C \times G(C).$$

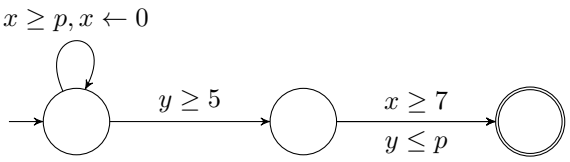


Figure 1: A parametric timed automaton with two clocks $\{x, y\}$ and a single parameter p. The final state is reachable if, for example, $p = 10$.

For each edge $e = (s, s')$ with $\lambda(e) = (R, G)$, the set R specifies which clocks are reset by taking the edge and G is the guard specifying for which clock values the edge is enabled.

A *configuration* (s, ν) of A consists of a state s and function $\nu : C \to T$ assigning a value from the time domain T to each clock. Timed automata can be interpreted either over dense time ($T = \mathbb{R}_{\geq 0}$) or over discrete time ($T = \mathbb{N}_{\geq 0}$).

The initial clock valuation ν_0 assigns 0 to every clock and the initial configuration is (s_0, ν_0).

An execution of a timed automaton is a combination of staying in the same state letting the time evolve and (instantaneous) state transition by taking an edge. Precisely, a transition exists from configuration (s, ν) to (s', ν') in A ,written $(s, \nu) \to (s', \nu')$, if either

- there exists $t \in T$ such that $s = s'$ and $\nu(c) + t = \nu'(c)$ for every clock $c \in C$
- or there is an edge $e = (s, s') \in E$ with $\lambda(e) = (R, G)$ such that G is satisfied for current clock values ν and $\nu'(c) = \begin{cases} 0 & \text{if } c \in R \\ \nu(c) & \text{if } c \notin R \end{cases}$

A *run* of a timed automaton is a sequence $\pi = \boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}$ of configurations such that $\boldsymbol{c_i} \to \boldsymbol{c_{i+1}}$ for each i. A run is called *accepting* if $\boldsymbol{c_1}$ is the initial configuration and $\boldsymbol{c_k}$ is in a final state. When clear from the context, we use runs to denote both sequences of configurations as well a the sequences of underlying edges.

For many practical consideration, requiring all guards in a timed automaton to be concretely specified is too restrictive. We now introduce parametric extension of timed automata which allows the constraints to be parametrised.

Formally, let P be a finite set of *parameters*. Then for the set of clocks C, the expression $G(C, P)$ denotes the set of guards (similarly to $G(C)$) where the clocks can be compared to constants as well as to parameters: $\bigwedge_i c \bowtie d$ where $c \in C$ is a clock, $\bowtie \in \{\leq, =, \geq\}$ and $d \in \mathbb{N} \cup P$.

Definition 4.2. *A* parametric timed automaton *A over the set of clocks C and parameters P is an automaton with*

$$Op = 2^C \times G(C, P).$$

An example of a parametric timed automaton is shown in Figure 1. By instantiating parameters, we obtain timed automata.

An *assignment* for P is a function $\gamma : P \to \mathbb{N}$ assigning a *natural number* to each parameter. Given a parametric timed automaton A and an assignment γ, the expression A^γ denotes the timed automaton obtained by instantiating every parameter $p \in P$ at $\gamma(p)$. The main problems studied in this manuscript is the problem of finding a parameter valuation such that a given parametric timed automaton has an accepting run, i.e., for some input values of parameters, is there a terminating computation for the parametric timed automaton? Formally,

Problem 4.3.

Name:	Existential Emptiness Problem
Input:	*Parametric Timed Automaton A*
Problem:	*Is there an assignment γ such that A^γ has an accepting run?*

The existential emptiness problem is also known in literature [2] as the parametric reachability or the emptiness problem for parametric timed automata. We omit "existential" in this manuscript and write

simply "emptiness problem". We say that two automata A_1 and A_2 have *equivalent emptiness problem* if the solution of the emptiness problem for A_1 is true if and only if the solution of the emptiness problem for A_2 i true.

4.2. One-Counter Machines

Counter machines are obtained from finite automata by adding counters. We deliberately use the word "machine" to describe automata extended with counters as this disambiguates presentation in sections 6 and 7 where we show a relationship between timed automata and counter machines and study the properties of the former using the latter.

Each counter of a counter machine ranges over natural numbers and each edge can increment, decrement or test for zero each of the counters independently. It is well-known [22] that having two counters makes the machine as computationally powerful as a Turing machine.

In this manuscript, we focus on problems related to reachability in (parametric) timed automata. We will prove that such problems are equivalent to reachability in certain classes of (parametric) counter machines with only a single counter.

Definition 4.4. *A* one-counter machine C *is an automaton with*

$$Op = \{+c, -c, =c, \geq c \ : \ c \in \mathbb{N}\}.$$

Unlike two-counter machine, the reachability problem for one-counter machines is decidable and in fact in NP [10] even when the constants are encoded in binary. A *configuration* (s, x) of C consists of a state $s \in S$ and counter value $x \in \mathbb{N}$. That is, the counter is always required to stay nonnegative. The initial configuration of C is $(s_0, 0)$. A configuration (s, x) is accepting if $s \in F$ is final state.

A configuration (s', x') is reachable in one step from (s, x) (written $(s, x) \to (s', x')$) in C if there exists an edge $e = (s, s') \in E$ such that

- if $\lambda(e) = \pm c$ then $x \pm c = x'$
- if $\lambda(e) = \sim c$ then $x = x'$ and $x \sim c$ where $\sim \in \{=, \geq\}$

A *r*un of a one-counter automaton is a sequence $\pi = \boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}$ of configurations such that $\boldsymbol{c_i} \to \boldsymbol{c_{i+1}}$ for each index i. A run is called *a*ccepting if $\boldsymbol{c_1}$ is the initial configuration and $\boldsymbol{c_k}$ is in a final state.

Notice that the '$\geq c$' edge is a syntactic sugar for a gadget consisting of a '$-c$' edge followed by a '$+c$' edge. To show equivalence between parametric timed automata and parametric one-counter machine we shall also require the other comparison '$\leq c$' edge. This is analogous to nonparametric settings where an equivalence was established previously [11].

A *simple bounded one-counter machine* M is obtained by extending the codomain Op of the function λ assigning an operation to each edge by '$\leq c$' operation for $c \in \mathbb{N}$.

Definition 4.5. *A* simple bounded one-counter machine C *is an automaton with*

$$Op = \{+c, -c, =c, \geq c, \leq c \ : \ c \in \mathbb{N}\}.$$

Configuration (s', x') is reachable in one step from configuration (s, x) using a '$\leq c$' transition

- if $\lambda(e) = \leq c$ then $x = x'$ and $x \leq c$

It was shown in [7] that allowing '$\leq c$' edges makes the reachability problem PSPACE-complete.

In order to show the reductions from parametric timed automata we also need two more types of operations: '$\equiv 0 \bmod c$' and '$+[0, c]$' for $c \in \mathbb{N}$. Configuration (s', x') is reachable in one step from configuration (s, x) using these edges provided it satisfies:

- $\lambda(e) = +[0, c]$ and $x \leq x' \leq x + c$
- $\lambda(e) = \equiv 0 \bmod c$ and $x = x'$ and $x \equiv 0 \bmod c$

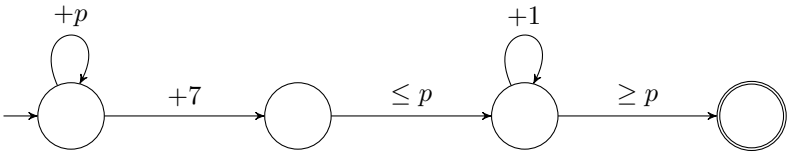


Figure 2: Example of a parametric bounded one-counter machine. The final state is reachable for all $p \in [7, \infty)$.

Definition 4.6. *A* bounded one-counter machine[1] *C is an automaton with*

$$Op = \{+c, -c, =c, \geq c, \leq c, +[0,c], \equiv 0 \mod c \ : \ c \in \mathbb{N}\}.$$

We use $\text{counter}(s, x) = x$ to denote the counter value in the configuration (s, x). We extend the definition to runs componentwise: if π is a run in C then $\text{counter}(\pi)$ is a sequence of natural numbers obtained by applying counter to each element of π. For a contant $K \in \mathbb{N}$, we write $\text{counter}(\pi) \leq K$ (resp. $\text{counter}(\pi) \geq K$) if the comparison holds for every element: $\forall i \,.\, \text{counter}(\pi(i)) \leq K$ (resp. $\forall i \,.\, \text{counter}(\pi(i)) \geq K$).

If the last configuration $\text{last}(\pi_1)$ of run π_1 is the same as the first configuration $\text{first}(\pi_2))$ of run π_2 then π_1, π_2 can be concatenated into a single run $\pi_1\pi_2$ in the obvious way.

Each run in C modifies the counter. By $\text{effect}(\pi)$ we denote the change in the counter: $\text{effect}(\pi) = \text{counter}(\text{last}(\pi)) - \text{counter}(\text{first}(\pi))$. A run π is called *positive* if $\text{effect}(\pi) > 0$. It is called *negative* if $\text{effect}(\pi) < 0$. Note that if π is a loop and $a \in \mathbb{N}$ then $\text{effect}(\pi^a) = a\,\text{effect}(\pi)$.

Further, for a run π we define $\text{div}(\pi) = \max_i |\,\text{counter}(\pi(i)) - \text{counter}(\text{first}(\pi))|$ to be the maximum difference between the counter and its initial value in the run π. If a run π can be written as $\pi = \pi_1\pi_2$ then it follows from the triangle inequality that $\text{div}(\pi) \leq \text{div}(\pi_1) + \text{div}(\pi_2)$.

A *path* in C is a path in the underlying graph. Given an initial configuration (s, x), a path π starting in a state s uniquely determines a run in C. The run follows edges as given by π and the counter value is updated accordingly. Thus, if the initial configuration is known, we often identify runs with paths and vice versa.

To reason about parametric timed automata, we introduce parametric one-counter machines. Formally, let P be a finite set of *parameters*. An *assignment* for P is a function $\gamma : P \to \mathbb{N}$ assigning a *natural number* to each parameter.

Definition 4.7. *A* parametric bounded one-counter machine *C over the set of parameters P is an automaton with*

$$Op = \{+c, -c, +p, -p, \leq c, = c, \geq c, \leq p, = p, \geq p, +[0,p], \equiv 0 \mod c \ : \ c \in \mathbb{N}, p \in P\}.$$

We deliberately omitted the parametric version of '$\equiv 0 \mod p$' as such an operation is not needed in the reduction from parametric timed automata to parametric bounded one-counter machines. Parametric version of simple one-counter machines is obtained in the analogous way.

Definition 4.8. *A* simple parametric bounded one-counter machine *C over the set of parameters P is an automaton with*

$$Op = \{+c, -c, +p, -p, \leq c, = c, \geq c, \leq p, = p, \geq p \ : \ c \in \mathbb{N}, p \in P\}.$$

Given a parametric one-counter machine C and an assignment γ the expression C^γ denotes the one-counter machine obtained by instantiating every parameter $p \in P$ at $\gamma(p)$. As for timed automaton, we have an equivalent notion of the emptiness problem for counter machines.

Problem 4.9.
Name: Existential Emptiness Problem
Input: *Parametric Bounded One-Counter Machine C*
Problem: *Is there an assignment γ such that C^γ has an accepting run?*

[1]Note that our definition is more general than the one given in [11].

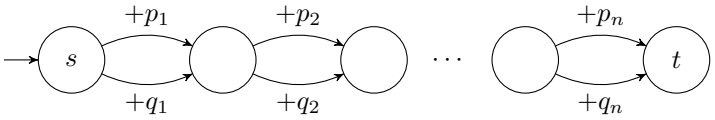


Figure 3: A parametric one-counter automaton.

5. Presburger Arithmetic

Presburger Arithmetic with Divisibility is the first-order logical theory over the structure $\langle \mathbb{N}, <, +, |, 0, 1 \rangle$ where $|$ denotes the standard divisibility operator on natural numbers. The existential fragment (formulae of the form $\exists x_1, x_2, \ldots, x_k.\varphi$ where φ has no quantifiers) is denoted as $\exists$PAD.

For example, the relation '$x \equiv y \bmod z$' can be expressed by a $\exists$PAD formula:

$$\varphi(x, y, z) := \exists p \,.\, (z|p \ \wedge \ x - p = y \ \wedge \ y < z).$$

The satisfiability of $\exists$PAD formulae was shown to be decidable in [19, 4] and recently to be in NEXP [18]. Unlike claimed in several previous works [5, 10, 9], the satisfiability of $\exists$PAD is not known to be in NP. At the time of the writing, the precise complexity of satisfiability of $\exists$PAD sentences is unknown; with the best lower bound NP-hard and the best upper bound NEXP. We refer the reader to the Related Work section and [18] for a more thorough discussion of the current state of $\exists$PAD satisfiability.

Given a set $S \subseteq \mathbb{N}^k$ we say that S is $\exists$PAD *definable* if there is a finite set R of $\exists$PAD formulae[2], each formula with free variables $y_1, \ldots y_k$ such that $(n_1, \ldots, n_k) \in S \iff \bigvee_{\varphi \in R} \varphi(n_1, \ldots, n_k)$. Note that $\exists$PAD sets are closed under union, intersection and projection.

It was shown in [9, 10] that the reachability relation of parametric one-counter machines is $\exists$PAD definable.

Lemma 5.1 ([9], Lemma 4.2.2)**.** *Given a parametric one-counter machine C (i.e., no upper bounds, '$+[0,p]$' or '$\equiv 0 \bmod c$' transitions) and states s, t, the relation* $\mathrm{Reach}(C, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \to^* (t, y)$ *in* C^γ *where* $\gamma(p_i) = n_i\}$ *is* $\exists$PAD *definable.*

6. Relationship Between Parametric Timed Automata and Parametric Bounded One-Counter Machines

Let us restate the main problem studied in this manuscript:

Name:	*Existential Emptiness Problem*
Input:	Parametric Timed Automaton A
Problem:	Is there an assignment γ such that A^γ has an accepting run?

As stated in the introduction and preliminaries the problem becomes undecidable for automata with three or more parametric clocks. In this section we show that given a parametric timed automaton with at most two parametric clocks there is a parametric bounded one-counter machine with equivalent emptiness problem. Then in the subsequent sections we show how to decide the emptiness problem for the latter class.

The convention used in this and the following section is that timed automata are denoted by A and counter machines are denoted by C. The following theorem shows that it suffices to restrict to runs over integer-time only.

Theorem 6.1. *Let A be a parametric timed automaton such that all constraints in C are closed (i.e., $c \leq d$, $c \geq d$) then the following are equivalent.*

[2] A single formula would be logically sufficient, but would result in exponential blowup. Instead, a $\exists$PAD decision procedure needs to only guess a formula in R and check its satisfiability

- *There is an assignment γ_1 such that there is an accepting run π_1 in A^{γ_1} over discrete time.*
- *There is an assignment γ_2 such that there is an accepting run π_2 in A^{γ_2} over dense time.*

Proof. Clearly, the integer-valued run π_1 is a run in A^{γ_1} also over dense time.

Conversely, suppose that there is such γ_2 and π_2. Then, A^{γ_2} is a closed timed automaton with an accepting run π_2. Hence, by [23], there is an integer-valued accepting run π_1 in A^{γ_2}. That is, the clock values in π_1 are always integers. But then π_1 is an accepting run in A^{γ_1} over discrete time. □

6.1. Nonparametric Clock Elimination

Let A be a parametric timed automaton. In this section we show, by modifying the region construction [3], how to build a parametric timed automaton with equivalent emptiness problem but without nonparametric clocks.

Note that once the value of a nonparametric clock c is above the largest constant appearing in A, the precise value of the clock does not affect any comparison. Since we restrict to discrete time, the value of c is always a natural number. We eliminate all nonparametric clocks by storing in the state space of A the values of clocks up to the largest constant. However, we must ensure that the eliminated clocks progress simultaneously with the remaining parametric clocks. This motivates parametric 0/1 timed automata where the $+1$ updates correspond to the progress of time whereas the $+0$ updates correspond to taking an edge in A.

Definition 6.2. *A* parametric 0/1 timed automaton *over the set of clocks C and parameters P is an automaton with*

$$Op = \{+0, +1\} \times 2^C \times G(C, P).$$

Formally, the following result appeared in [2].

Lemma 6.3 ([2])**.** *Let $A = (S, s_0, C, P, F, E, \lambda)$ be a parametric timed automaton. Then there is a parametric 0/1 timed automaton $A' = (S', s'_0, C', P', F', E', \lambda')$ such that $C' \subseteq C$ contains only parametrically constrained clocks of C and A and A' have equivalent emptiness problem. Moreover, $|A'| = O(2^{|A|})$.*

6.2. One Parametric Clock

In case the original parametric timed automaton A uses only one parametric clock we show that the corresponding parametric 0/1 timed automaton has a simple structure.

For the rest of the section, fix a parametric timed automaton A with one parametric clock. By Lemma 6.3, there is an exponentially larger parametric 0/1 automaton B with one (parametrically constrained) clock such that A and B have equivalent emptiness problem.

Now, B has a single clock which is incremented by 0 or 1 in each transition. So B is almost a parametric bounded one-counter machine, only that B may contain clock resets. However, clock resets can be eliminated from B by introducing -1 transitions.

Lemma 6.4. *Let B be a parametric 0/1 timed one-clock automaton. Then there is a parametric bounded one-counter machine C such that B and C have equivalent emptiness problem. Further, all updates in C are either $-1, 0$ or $+1$ and $|C| = O(|B|)$.*

Proof. Counter machine C has the same states and $+0, +1$ transitions as B. By modifying B if necessary, we can assume that all edges resetting a clock are $+0$ edges without any comparison (Automata constructed in Lemma 6.3 have this property). Each transition resetting a clock is then replaced by a gadget (see Figure 4) that subtracts 1 from the counter until the counter equals 0. □

Hence, to decide the emptiness problem for A it suffices to decide the emptiness problem for a parametric bounded one-counter machine with only $-1, 0, +1$ counter updates. We establish such a result in Theorem 8.6, which then yields:

Theorem 6.5. *The emptiness problem for parametric timed automata with one parametric clock is decidable in* 2NEXP *time.*

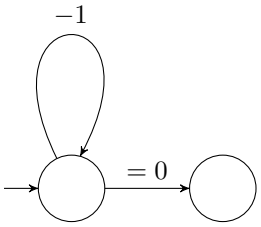


Figure 4: Gadget implementing a clock reset.

Decidability of the emptiness problem for the above class of timed automata was first given in [2], albeit with nonelementary complexity. Our proof is different, uses one-counter machines and yields a 2NEXP algorithm.

For the lower bound, we show that the problem is at least NEXP-hard. Note that the upper bound proof works by reduction to the satisfiability of (exponentially large) ∃PAD formula. The complexity of the latter problem is still open [18] and lies between NP-hard and NEXP. Hence, depending on the results on the ∃PAD satisfiability, the upper bound can lie between NEXP and 2NEXP. Therefore, depending on the developments in ∃PAD, the NEXP lower bound might be optimal and the emptiness problem for parametric timed automata with one parametric clock can still be solvable in NEXP.

Alternatively, strengthening the lower bound of the emptiness problem to 2NEXP would entail that ∃PAD is not in NP (Unless NEXP and 2NEXP coincide).

Theorem 6.6. *The emptiness problem for parametric timed automata with one parametric clock and one parameter is NEXP-hard.*

Proof. The proof is by reduction from SuccinctSAT, a well-known NEXP-complete problem. A similar construction was used in [8] to show that LTL model checking of parametric one-counter machines is NEXP-hard.

The SuccinctSAT problem is the adaptation of the classical SAT problem in which the input formula is not given explicitly but instead is encoded by a Boolean circuit $\mathbb{C}$. If $\mathbb{C}$ has k inputs, we require that the size of $\mathbb{C}$ is polynomial in k. Such circuits can encode exponentially larger formulas as follows.

We assume that the input and the output of the circuit $\mathbb{C}$ are bit strings—the values on the input and the output wires. We denote the concatenation of bit strings u and v by $u \cdot v$.

Let b be an n bit number and $i \in \{0, 1, 2\}$. The circuit operates in such a way that evaluating the circuit $\mathbb{C}(b \cdot i) = v \cdot w$ gives the index of the variable $v = f(b, i)$ in the ith literal in the bth clause. Moreover, if the literal is negated then $w = 1$, otherwise, $w = 0$.

Let φ be a 3SAT formula encoded by circuit $\mathbb{C}$. The variables of φ are denoted by $x_1, x_2, \ldots$. Indexing the clauses of φ by n bit strings, we thus write φ as

$$\varphi = \bigwedge_{b \in \mathbb{B}^n} \Big[(\neg) x_{f(b,0)} \vee (\neg) x_{f(b,1)} \vee (\neg) x_{f(b,2)} \Big].$$

where $f(b, i)$ is the index of the ith variable in the bth clause.

We shall construct a parametric timed automaton A with one parametric clock using a single parameter p such that a final state of A is reachable for some value of p if and only if the value of p encodes a satisfying assignment for φ.

An assignment $v : \{x_1, x_2, \ldots\} \to \{0, 1\}$ for φ is encoded in p as follows. Let π_k be the k-th prime.

$$v(x_k) = \begin{cases} 1 & \text{if } \pi_k | p \\ 0 & \text{if } \pi_k \nmid p \end{cases}$$

That is, $(v(x_k) = 1) \iff \pi_k | p$. By choosing distinct primes as the basis of the encoding, any assignment to variables in φ can be represented by a number and vice versa.

Algorithm 1 Algorithm iterating over all clauses of φ and checking that each is satisfiable. $\mathbb{S}_{divides}(p, x)$ stands for subroutine calculating whether x divides p and $x \leftarrow \mathbb{S}_{n^{th}prime}(v)$ assigns the vth prime to x.

for $b \leftarrow 0$ **to** 2^n **do** ▷ Iterate over all clauses
  $s \leftarrow$ false ▷ Denotes whether some literal evaluates to true
  for $i \leftarrow 0$ **to** 2 **do** ▷ Try all three literals
    $v \cdot w \leftarrow \mathbb{C}(b \cdot i)$
    $r \leftarrow \mathbb{S}_{n^{th}prime}(v)$ ▷ Calculate the corresponding prime number
    if $w = 0 \wedge \mathbb{S}_{divides}(p, r)$ **then**
      $s \leftarrow$ true
    end if
    if $w = 1 \wedge \mathbb{S}_{notdivides}(p, r)$ **then**
      $s \leftarrow$ true
    end if
  end for
  if $s =$ false **then** ▷ If the current clause is not satisfied under the assignment; reject
    reject
  end if
end for
accept ▷ All clauses are satisfies; accept

The timed automaton A then implements the algorithm that iterates over all clauses and checks that each clause is satisfiable under the assignment encoded in p (see Algorithm 1). Hence, φ is satisfiable if and only if some value of p encodes a satisfying assignment which holds if and only if a final state of A is reachable.

To implement the algorithm using a parametric timed automaton, we provide various gadgets: a gadget to calculate the kth prime number and gadgets to check (non)divisibility of p by the calculated prime numbers.

Note that the problem of calculating the kth prime number is in PSPACE. Now, it is well known that the reachability for nonparametric timed automata is PSPACE-complete [3] and so many different ways exist of using timed automata to calculate PSPACE functions. A particular approach is sketched below.

Since A has no restrictions on the number of nonparametric clocks, we use two nonparametric clocks x_a, x_b to store a single bit; interpreting the bit as 1 when $x_a = x_b$ and as 0 otherwise. Note that clock comparisons and resets can be used to check and set the bit, respectively[3]. We use nonparametric clocks as a memory. Now, A uses polynomially many clocks to store polynomially many bits. It is easy to see that using the finite state control of A and polynomially many bits, parametric timed automaton A can calculate any PSPACE function. Similarly, we use n bits to iterate over all possible values of b in the algorithm.

In our implementation of the algorithm in Algorithm 1, we use m bits to store the kth prime number in binary. Further, the algorithm uses subroutines to check whether p is divisible by a given number. We use the gadget shown in Figure 5 to check (non)divisibility of p by prime numbers. Notice that the gadgets and hence the entire proof uses only a single parameter p. □

6.3. Two Parametric Clocks

We now move to parametric timed automata with two parametric clocks. Decidability in this case in general is open since the introduction of the problem [2] over two decades ago. This is analogous to reachability in nonparametric timed automata where the precise complexity in the two-clock case was a major open problem until recently shown to be PSPACE-complete [7] (as opposed to NL-complete for one clock [17]).

[3]Alternatively, resetting clocks x_a, x_b whenever they reach 1 and checking for the clocks being simultaneously 0 or 1, we can implement bits without direct comparison of two clocks.

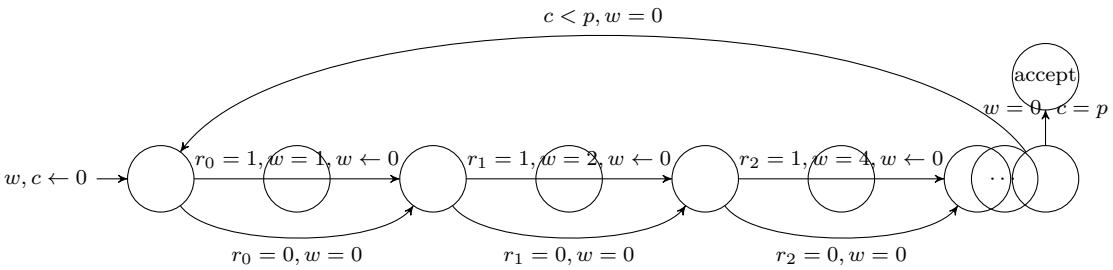


Figure 5: Parametric timed automaton implementing r divides p. The bits $r_0, r_1, r_2, \ldots, r_m$ represent $r = \sum_i r_i 2^i$ in binary. An auxiliary clock w counts to the powers of 2. The gadget is entered on the left. Note that one traversal through the gadget takes r time units. Clock c measures p. Note that the transition to the accepting state can be taken only if r divides p. Gadget checking r does not divide p is obtained by changing the final test $w = 0, c = p$ to $w = 0, c > p$.

Similarly, for counter machines, the reachability is NP-complete for one counter [10] but undecidable for two counters [22].

In this section we show that the decidability of the emptiness problem for parametric timed automata with two parametric clocks is equivalent to the emptiness problem for parametric bounded one-counter machines. The difference from the results in the previous section is that the counter can be compared as well as incremented/decremented by a parameter.

The equivalence is then used in Section 9 to show decidability of the emptiness problem in case the parametric timed automaton uses only a single parameter.

First, observe that a counter can be stored as a difference of two clocks, which can be used to show the easier direction of the equivalence.

Theorem 6.7. *Let C be a parametric bounded one-counter machine. Then there is a parametric timed automaton A with two parametric clocks such that A and C have equivalent emptiness problem. If C has no '$\equiv 0 \bmod c$' transitions then A has no nonparametric clocks. Otherwise, A has one nonparametric clock. Moreover, A is constructible in* P.

Proof. A similar proof appeared in [11]. Timed automaton A tracks the state of C and the counter value of C is stored as the difference $x - y$ of two clocks. Since the clocks progress simultaneously, the difference is constant.

Parametric timed automaton A has the same set of parameters as C together with a fresh parameter M. Intuitively, M is an upper bound on counter values in an accepting run of C. Automaton A ensures that whenever the clock x or y reaches M the clock is reset.

Resetting clocks when they reach M allows us to implement counter operations in A. For example, an update $+p$ can be implemented by resetting y when $y = p$. The test $\leq p$ can be implemented by checking $y = 0 \wedge x \leq p$. And the update '$+[0, p]$' can be implemented by resetting y when $y \leq p$. All other counter updates and comparisons can be implemented similarly.

To check '$\equiv 0 \bmod c$' we introduce a fresh clock z (see Figure 6). Clock z resets together with x when $x = M$. Thus, clocks x and z are zero at the same time. Then every time z reaches c, the clock z is reset. Therefore, z counts modulo c. So '$\equiv 0 \bmod c$' reduces to checking that when $y = M$ the value of z equals 0.

Note that since at any single time we make at most one '$\equiv 0 \bmod c$' check, the clock z can be reused in different gadgets for '$\equiv 0 \bmod c$' checks for different constants. Hence only one clock suffices for all '$\equiv 0 \bmod c$' checks.

Finally, observe that x and y are parametrically constrained (by M and parameters already in C) whereas z is not. □

6.4. Reduction to Parametric Bounded One-Counter Machines

For the converse, fix A to be a parametric timed automaton with two parametric clocks. We reduce A to a parametric bounded one-counter machine C with equivalent emptiness problem. As the first step, we

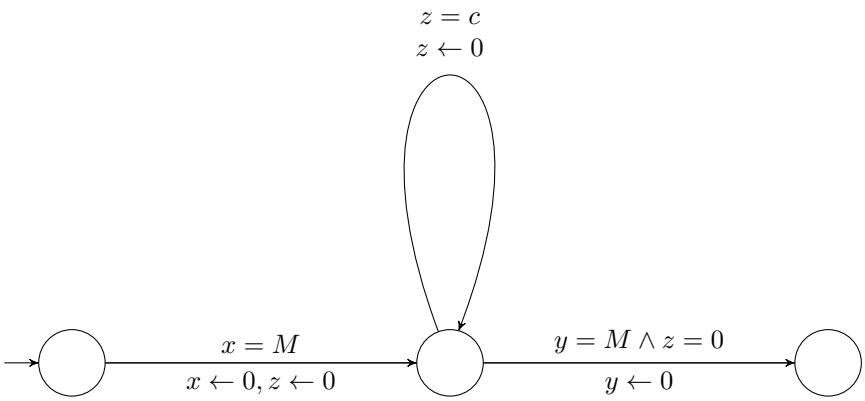


Figure 6: Gadget testing that the value stored as $x - y$ is divisible by the given constant $c \in \mathbb{N}$.

construct (Lemma 6.3) a parametric 0/1 timed automaton B with two parametrically constrained clocks, denoted x and y, with the emptiness problem equivalent to A. We now transform B to C.

Denote the counter of C by z. For the time being, we need to relax the assumption that z stays nonnegative. That is, subtracting 5 when the counter is 2 results in the counter being -3. In Remark 6.9 we later show how to restore the nonnegativity of the counter.

The idea of the reduction is that, after a clock of B is reset, that clock equals zero and we use z to store the value of the other clock. We construct C in such a way that after a reset of y, counter z stores the value of x and after a reset of x, counter z stores $-y$. Initially C starts with the counter equal to 0.

Machine C then operates in phases. Each phase corresponds to a run of B between two consecutive resets of some (possibly different) clock.

Suppose y was the last clock to reset. After the reset, the configuration of B is $(s, (z, 0))$ for some state $s \in B$ and the counter $z = x$. We show how C calculates the configuration after the next clock reset in B.

After time Δ, the clocks go from configuration $(z, 0)$ to $(z + \Delta, \Delta)$. Based on the guards, different transitions in B^γ are enabled as time progresses. For the time being, we assume that the order of the parameters is $p_1 < p_2 < \ldots < p_k$. Later we relax this assumption by building a separate automaton for each possible permutation.

Let *region* $R_{(i,j)}$ be the set of clock valuations $[p_i, p_{i+1}] \times [p_j, p_{j+1}]$. Then the set of enabled transitions depends only on the region $R_{(i,j)}$ the clocks (x, y) lie in.[4]

Therefore, machine C guesses (see Figure 7):

- the regions $R_{(i_0,j_0)}, R_{(i_1,j_1)}, \ldots, R_{(i_m,j_m)}$ in the order in which they are visited by the clocks (x, y),
- the states $s_0, s_1, \ldots, s_m$ of B when each region R_l is visited for the first time,
- the state t of B in which the next reset occurs,
- which clock is reset next.

Machine C then checks that the sequence is valid. First, C checks, that $(z, 0)$ lies in R_0. Second, it checks that the regions are adjacent:

$$(i_{l+1} - i_l = 1 \wedge j_{l+1} = j_l) \text{ or } (i_{l+1} = i_l \wedge j_{l+1} - j_l = 1) \text{ or } (i_{l+1} - i_l = j_{l+1} - j_l = 1).$$

The last case corresponds to the clocks hitting a corner of a region. Note that this forces $m \leq 2k$. Then, C checks that starting in clock configuration $(z, 0)$, the regions can be visited in the guessed order.

[4]Our definition of rectangular regions differs slightly from the one usually given in the literature [3]. However, as all inequalities are nonstrict the regions are sufficient. For ease of presentation, we also use the convention $p_0 = 0$ and $p_{k+1} = \infty$.

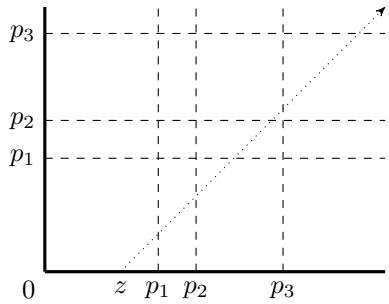


Figure 7: Regions for three parameters $p_1 < p_2 < p_3$. The dotted line shows an evolution of clock configuration, which visits $R_{(0,0)}, R_{(1,0)}, R_{(2,0)}, R_{(2,1)}, R_{(2,2)}, R_{(3,2)}, R_{(3,3)}$.

Consider region $R_{(u,v)}$ for some u, v. When the region is visited for the first time, then either clock x equals p_u or clock y equals p_v. In the former case, the clock configuration of B is $(p_u, p_u - z)$, in the latter case, it is $(p_v + z, p_v)$. The configuration depends on the direction in which $R_{(u,v)}$ is visited. See Figure 7.

- If $i_{l+1} - i_l = 1$ then C checks that clock x reaches $p_{i_{l+1}}$ before clock y reaches p_{j_l+1}. That is, $p_{i_{l+1}} - z \leq p_{j_l+1}$. Equivalently, $p_{i_{l+1}} \leq z + p_{j_l+1}$, which can be easily tested by a parametric bounded one-counter machine[5]. In Figure 7 this corresponds to region $R_{(1,0)}$, which is visited before $R_{(2,0)}$.
- Similarly, if $j_{l+1} - j_l = 1$ then C checks that clock y reaches $p_{j_{l+1}}$ before clock x reaches p_{i_l+1}. That is, $p_{i_l+1} - z \geq p_{j_{l+1}} - z$. Equivalently, $p_{i_l+1} \geq p_{j_{l+1}} + z$, which can be easily tested by a parametric bounded one-counter machine. In Figure 7 region $R_{(2,1)}$ is visited before $R_{(2,2)}$.

We say that R_l was *reached from left* in the first and that R_l was *reached from bottom* in the second case. The notation R_l is the shorthand for the lth visited block $R_{(i_l,j_l)}$. See Figure 7 for the intuition behind the names.

Finally, C checks reachability within individual regions. Let $\boldsymbol{c_l}$ be the configuration in which the region R_l is visited for the first time. Then C checks that a run from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ exists in R_l.

Now, with each $R_{(i,j)}$, we introduce a one-counter machine $B_{(i,j)}$ obtained from B assuming clock $x \in [p_i, p_{i+1}]$ and clock $y \in [p_j, p_{j+1}]$, instantiating all comparisons accordingly and by removing all edges resetting a clock.

Formally, for each i and j, the automaton $B_{(i,j)}$ is obtained from B by

- removing all edges resetting a clock,
- removing all edges labelled by $x \geq p_k$ for $k > i$,
- removing all edges labelled by $x \leq p_k$ for $k \leq i$,
- replacing by an $+0$ edge all edges labelled by $x \geq p_k$ for $k \leq i$,
- replacing by an $+0$ edge all edges labelled by $x \leq p_k$ for $k > i$,
- removing all edges labelled by $y \geq p_k$ for $k > j$,
- removing all edges labelled by $y \leq p_k$ for $k \leq j$,
- replacing by an $+0$ edge all edges labelled by $y \geq p_k$ for $k \leq j$,
- replacing by an $+0$ edge all edges labelled by $y \leq p_k$ for $k > j$,

Notice that $B_{(i,j)}$'s are 0/1 automata without resets, comparisons or parameters, i.e, one-counter machines. In particular, the reachability relation for $B_{(i,j)}$'s is semilinear. Formally, for a pair of states s and t of a one-counter machine X define $\Pi(X, s, t)$ to be the set of counter values reachable at t by a run starting in state s and counter equal to 0: $\Pi(X, s, t) = \{v \mid \exists \pi \in X \,.\, \text{first}(\pi) = (s, 0) \wedge \text{last}(\pi) = (t, v)\}$.

Lemma 6.8. *Let X be a one-counter machine with 0/1 updates. Then for any states $s, t \in X$ the set $\Pi(s, t)$ is effectively semilinear: $\Pi(X, s, t) = D \cup \bigcup_{1 \leq j \leq r} \{a_j + b_j \mathbb{N}\}$ where $D \subseteq \mathbb{N}$ is finite and $a_j, b_j \in \mathbb{N}$.*

[5]The test can be implemented by the sequence of the following edges: $+p_{j_l+1}, \geq p_{i_{l+1}}, -p_{j_l+1}$

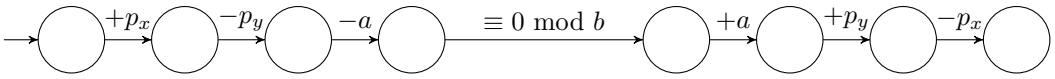


Figure 8: Gadget testing that for given $a, b \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that $z + p_x - p_y = a + kb$, i.e., $z + p_x - p_y - a \equiv 0 \bmod b$. Letter z denotes the current counter value.

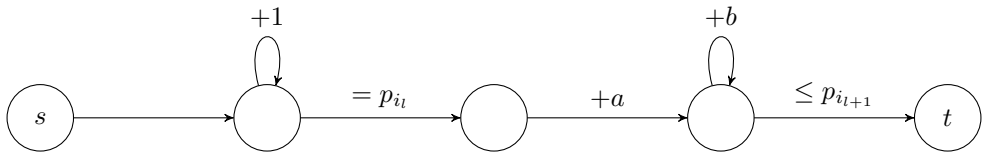


Figure 9: A gadget implementing a reset of clock y in the left/left situation.

Proof. It was shown in [9], Lemma 4.1.18, that the reachability relation in nonparametric one-counter machines is definable in the existential fragment of Presburger arithmetic (without divisibility). It is well known that this fragment defines effectively semilinear sets. □

Edges in $B_{(i,j)}$ never decrease the clock and hence the restriction to paths starting with the counter equal to 0 in the definition of $\Pi(X, s, t)$ is sufficient.

Now, to check that a run from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ exists in R_l, machine C distinguishes whether R_l and R_{l+1} are reached from bottom or from left and uses the semilinearity of the reachability relation of the corresponding $B_{(i,j)}$.

The translation is technical. For example, suppose $R_l = R_{(p_x,p_y)}$ for some parameters p_x and p_y. Then $\boldsymbol{c_l} = (s_l, (p_x, p_x - z))$ or $\boldsymbol{c_l} = (s_l, (p_y + z, p_y))$ depending on the direction. If R_l was reached from left and R_{l+1} from bottom then C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $z + p_{y+1} - p_x \in \Pi(B_l, s_l, s_{l+1})$. This and all other semilinear constraints can be checked by C using '$\equiv 0 \bmod c$' transitions (cf. Figure 8).

We now explain how to handle the remaining cases. Consider region $R_l = R_{(p_x,p_y)}$ for some parameters p_x and p_y, i.e., $R_l = (p_x, p_{x+1}) \times (p_y, p_{y+1})$. Then, R_l is visited for the first time in configuration $\boldsymbol{c_l} = (s_l, (p_x, p_x - z))$ or $\boldsymbol{c_l} = (s_l, (p_y + z, p_y))$. Then C checks that it is possible to go from $\boldsymbol{c_l}$ to $\boldsymbol{c_{l+1}}$ in R_l. The check distinguishes whether R_l and R_{l+1} were reached from bottom or from left.

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, C has to check that $(s_{l+1}, (p_{x+1}, p_{x+1} - z))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $p_{x+1} - p_x \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C (similarly as in Figure 8).
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_x, p_x - z))$. That is, that $p_{y+1} + z - p_x \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C (Figure 8).
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, C has to check that $(s_{l+1}, (p_{x+1}, p_{x+1} - z))$ is reachable from $(s_l, (p_y + z, p_y))$. That is, that $p_{x+1} - p_y - z \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C.
- R_l **reached from bottom,** R_{l+1} **reached from bottom**. Thus, C has to check that $(s_{l+1}, (p_{y+1} + z, p_{y+1}))$ is reachable from $(s_l, (p_y + z, p_y))$. That is, that $p_{y+1} + z - p_y - z = p_{y+1} - p_y \in \Pi(B_l, s_l, s_{l+1})$, which can be checked by C.

Finally, we show how to simulate by C a clock reset in B. Suppose that instead of resetting the clock, we let the time evolve. Then eventually the clock configuration transitions to another region. Denote that region by R_{l+1}.

First, suppose that y is the next clock to be reset. Then the first thing to check is whether there is a transition leaving t that resets the clock y.

Simulation of a clock reset distinguishes whether R_l and R_{l+1} were reached from bottom or from left. By Lemma 6.8, the set $\Pi(B_l, s_l, t)$ is semilinear. So, C guesses a generator (a, b) in $\Pi(B_l, s_l, t)$ that it will use to reach t. Then C checks whether state t can be reached using that generator. There are four cases to consider depending on the direction from which R_l and R_{l+1} are reached. In all four cases, automaton C nondeterministically chooses the counter value when t is reached.

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, B reaches R_l in the configuration $(s_l, (p_x, p_x - z))$. Since R_{l+1} would be reached from left, machine B resets when clock $x < p_{x+1}$. So C has to set the counter to a value $p_x + \Pi(B_l, s_l, t) \in [p_x, p_{x+1}]$. To do that, C first sets the counter to p_x. Then C adds a to the counter and starts nondeterministically incrementing the counter by b, checking that the counter stays below p_{x+1}. See Figure 9.
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in the configuration $(s_l, (p_x, p_x - z))$. And B resets when clock $x < p_{y+1} + z$. Thus, the new counter value $z' = p_x + a + kb \leq z + p_{y+1}$. So C increments the counter by p_{y+1}. Then, it keeps subtracting 1 from the counter checking that it is at least $\geq p_x + a$. Then C nondeterministically terminates when it holds that $z' - p_x - a \equiv 0 \mod b$.
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, B reaches R_l in the configuration $(s_l, (p_y + z, p_y))$. And B resets when clock $x < p_{x+1}$. Now, C first increments the counter to p_y. Second, it adds a to the counter and starts nondeterministically incrementing the counter by b, checking that the counter stays below p_{x+1}.
- R_l **reached from bottom,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in the configuration $(s_l, (p_y + z, p_y))$. And B resets when clock $x < p_{y+1} + z$. Thus C increments the counter by a number in the range $p_y + [0, p_{y+1} - p_y]$ of the form $a + kb$. This can be achieved, by first calculating $z \mod b$ using the '$\equiv 0 \mod b$' transitions, then incrementing the counter using $+[0, p_{y-1} - p_y]$ and then checking that the new counter value z' satisfies $z' \equiv z + a \mod b$.

This finishes a reset of y. The case of resetting x can be handled analogously. Notice that once the value of a clock becomes larger than p_k its exact value is irrelevant to any future comparison. Hence, C tracks x and y only up to p_k and remembers which clocks exceed it. Hence, we can assume that the counter of C is always inside $[-p_k, p_k]$.

Next, we modify C to ensure that the counter is always nonnegative (and inside $[0, 2p_k]$). See Figure 10 for the automaton we build. Let C' be obtained from C by adding a new initial state and a $+p_k$ edge from the new to the original initial state. Further, any comparison edge (s, G, t) (e.g., where G is $\leq p_i$) is replaced by a gadget of three edges $(s, -p_k, q), (q, G, q')$ and $(q', +p_k, t)$ which subtract p_k from the counter, perform the original check and then add p_k to the counter thereby offsetting the counter by p_k.

Remark 6.9. *We can assume that the counter of C is always inside $[0, 2p_k]$.*

Note that the construction depends on the order of parameters. However, we can build an automaton for every possible order of parameters. Then check the order of parameters and transition into the automaton for the appropriate order (see Figure 10).

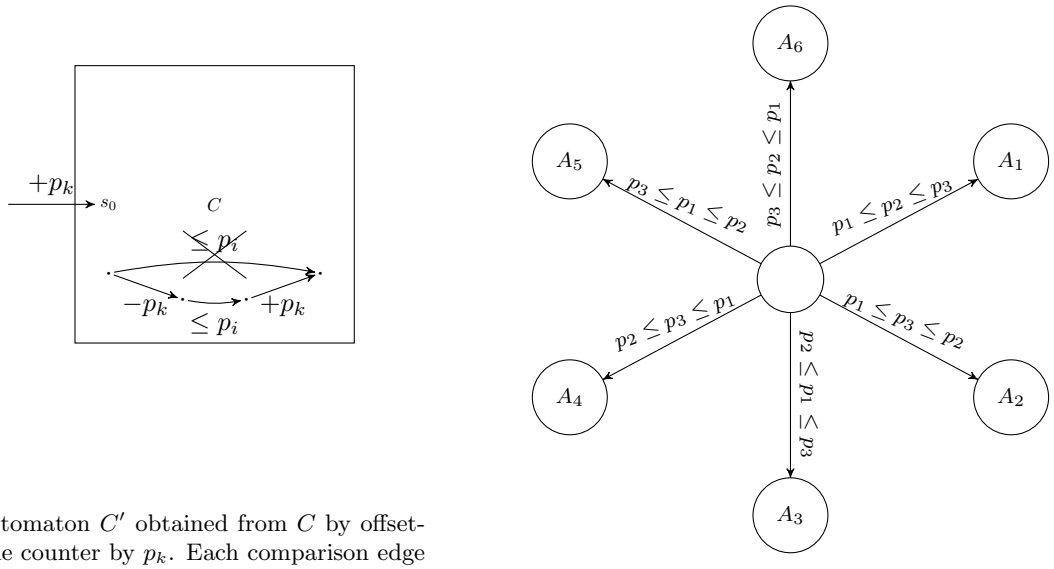
Finally, note that all '$\equiv 0 \mod c$' transitions are nonparametric. That is, $c \in \mathbb{N}$ is always a natural number and never a parameter.

Theorem 6.10. *For a parametric timed automaton A with two parametric clocks we can compute a parametric bounded one-counter machine C with equivalent emptiness problem.*

Our reduction was inspired by the work in [11]. Performing one phase in a single stage of C and using semilinearity of reachability in individual regions are the main differences from the reduction of [11].

Already for a single parameter, we have a $\mathsf{PSPACE}^{\mathsf{NEXP}}$ lower bound. Note that $\mathsf{PSPACE}^{\mathsf{NEXP}} = \mathsf{P}^{\mathsf{NEXP}}$ [1], however, we give a more involved proof showing $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hardness as opposed to $\mathsf{P}^{\mathsf{NEXP}}$ as we believe that the construction can be useful in improving the lower bound further.

Theorem 6.11. *The emptiness problem for parametric timed automata with two parametric clocks and a single parameter is $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hard.*



(a) Automaton C' obtained from C by offsetting the counter by p_k. Each comparison edge is replaced by three edges which subtract p_k from the counter, perform the original check and then add p_k to the counter.

(b) Parametric bounded one-counter automaton checking the order of parameters and transitioning into a corresponding subautomaton.

Figure 10: Gadgets used in the last step of the reduction from parametric timed automata to parametric one-counter machine

Proof. We first describe the $\mathsf{PSPACE}^{\mathrm{NEXP}}$-hard problem we reduce from. Recall from the proof of Lemma 6.6 that SuccinctSAT, whereby a 3SAT formula is represented by a Boolean circuit, is a NEXP-complete problem. So consider a PSPACE Turing Machine T that makes NEXP oracle calls by means of SuccinctSAT. Precisely, T is allowed to generate polynomial-size circuits encoding 3SAT formulae and then pass them to a SuccinctSAT oracle. It is clear that such machines can solve $\mathsf{PSPACE}^{\mathsf{NEXP}}$ problems.

Given such a machine T, we now describe how to build in polynomial time a parametric timed automaton with two parametric clocks simulating T. As the first step, we build a polynomial-size parametric timed automaton A with two parametric clocks deciding SuccinctSAT. This automaton is then used inside a nonparametric timed automaton as a NEXP oracle. Hence, the $\mathsf{PSPACE}^{\mathsf{NEXP}}$-hardness follows.

The construction of A is an extension of ideas from Lemma 6.6 for building a parametric timed automaton with one parametric clock. Recall that in the lemma we used a circuit $\mathbb{C}$ to encode a 3SAT formula φ. Further recall that in the lemma, we described how to use nonparametric clocks as memory. Automaton A uses polynomially many clocks to represent polynomial-size memory. The input to A is the description of circuit $\mathbb{C}$ encoded in this polynomial-size memory.

If $\mathbb{C}$ has $n+2$ inputs then φ can have as many as $3 \cdot 2^n$ variables, which we denoted by $x_1, x_2, \ldots$. Further recall that we represented an assignment v to variables in φ by a number $Z \in \mathbb{N}$ such that $v(x_k) = 1$ if and only if $\pi_k | Z$ where π_k is the kth prime.

Now, $\pi_k \leq 2k^2$. Hence, any valid assignment is represented by a number at most $lcm(1, 2, 3, \ldots, 2(3{\cdot}2^n)^2)$. On the other hand, $\pi_k \geq k$ and so some assignments are represented only by numbers greater than $(3 \cdot 2^n)!$, which is doubly exponential in n. Thus, storing Z directly would require exponentially many bits.

Let M be a parameter and suppose that $M \geq lcm(1, 2, 3, \ldots, 2(3 \cdot 2^n)^2)$. Then instead of using exponentially many bits, automaton A stores the value of an assignment as a difference $x - y$ of two parametric clocks x and y. The clocks operate modulo M. That is, whenever a clock reaches value M, the clock is reset. With this convention, the assignment stored by clocks x and y is the value of x when y equal 0. Notice that with this convention, resetting y when $y = 1$ increments the value of the stored assignment by one. The resetting trick is used by A to iterate over all assignments. The initial values of x and y, and hence the assignment, is 0.

Automaton A then checks whether an individual assignment satisfies φ similarly as is done in Lemma 6.6.

Algorithm 2 Algorithm iterating over all valid assignments and for each assignment iterating over all clauses of φ and checking that each is satisfiable. The automaton accepts if the formula is satisfiable and rejects otherwise.

for $i \leftarrow 1$ **to** $2(3 \cdot 2^n)^2$ **do**
  if $\mathbb{S}_{notdivides}(M, i)$ **then** ▷ Checks that $lcm(1, \ldots, 2(3 \cdot 2^n)^2) | p$
    reject
  end if
end for
$ass \leftarrow 0$ ▷ Stores the value of the assignment
while $ass \neq M$ **do** ▷ Try all possible assignments
  for $b \leftarrow 0$ **to** 2^n **do** ▷ Iterate over all clauses
    $ok \leftarrow$ true ▷ Denotes whether the current assignment satisfies φ
    $s \leftarrow$ false
    for $i \leftarrow 0$ **to** 2 **do** ▷ Check if some literal is true under the current assignment
      $v \cdot w \leftarrow \mathbb{C}(b \cdot i)$
      $r \leftarrow \mathbb{S}_{n^{th}prime}(v)$
      if $w = 0 \ \wedge \ \mathbb{S}_{divides}(M, r)$ **then**
        $s \leftarrow$ true
      end if
      if $w = 1 \ \wedge \ \mathbb{S}_{notdivides}(M, r)$ **then**
        $s \leftarrow$ true
      end if
    end for
    if $s =$ false **then**
      $ok \leftarrow$ false
    end if
  end for
  if $ok =$ true **then** ▷ Accept if found a satisfying assignment
    accept
  end if
  $ass \leftarrow ass + 1$
end while
reject ▷ No assignment satisfies φ; reject

Automaton A implements the Algorithm 2 that iterates over all clauses and checks that each is satisfied by the assignment. In order to check that an individual assignment satisfies φ, automaton A needs to be able to extract the value of each variable from the assignment. That is, A needs to be able to calculate the value of the kth prime π_k and to check whether π_k divides the value of the assignment. In Lemma 6.6 we described how A can calculate and represent π_k in binary $\pi_k = \sum_i r_i 2^i$ where r_i are bits.

Then checking whether π_k divides the assignment is done by modifying the gadget from Figure 5 used for the divisibility in Lemma 6.6. We modify the gadget so that it can be entered only when x is reset and it can be exited only when y is reset. This guarantees that exactly $x - y$ timeunits elapse during the traversal of the gadget.

This way, timed automaton A can iterate over all assignments for φ and check if at least one makes the formula true. Finally, to ensure that M is big enough, the automaton iterates over all numbers $k \in \{1, 2, 3, \ldots, 2(3 \cdot 2^n)^2\}$ and checks that $k|M$. This is possible, without using any additional parameters, as only polynomially many bits (clocks) are needed to store the value of each possible $k = 1, \ldots, 2(3 \cdot 2^n)^2$. The algorithm that is hard-wired into A is described in Algorithm 2.

Notice that A uses only two parametric clocks (x and y). Further notice that the only input to the algorithm is the circuit $\mathbb{C}$. The function f which takes an encoding of a circuit $\mathbb{C}$, input x and returns $f(\mathbb{C}, x) = \mathbb{C}(x)$ the value of $\mathbb{C}$ on x, is PSPACE computable. So we can modify the algorithm in Algorithm 2 to take an encoding of a circuit and accept if and only if the corresponding formula is satisfiable. Since SuccinctSAT is NEXP-complete, the algorithm can be used as a general NEXP oracle. The resulting automaton is constructible in polynomial time and is of polynomial size in the input. Thus, we can use it as an oracle in a PSPACE algorithm.

So the parametric timed automaton B simulating T works as follows. Automaton B uses polynomially many bits (clocks) to simulate T, making the same transitions as T does. Then whenever T makes an oracle call, automaton B resets x and y and then starts executing parametric timed automaton A on the circuit encoding the corresponding 3SAT formula.

Hence, there is a reduction from $\mathsf{PSPACE}^{\mathsf{NEXP}}$ problems to the emptiness problem for parametric timed automata with two parametric clocks. Observe that the value of the parameter M can be reused between the "oracle" calls, as M is an upper bound on the largest valid assignment encoded by a circuit with n inputs. □

7. Decidability Results for Parametric Bounded One-Counter Machines

We gave a reduction from the emptiness problem for parametric timed automata with at most two parametric clocks to the emptiness problem for parametric bounded one-counter machines. In this section we study the decidability of the reachability in the resulting one-counter machines. We use the equivalence to derive a 2NEXP algorithm for parametric timed automata with a single parametric clock, a decision procedure for parametric timed automata with two parametric clocks and a single parameter, and a decision procedure for parametric bounded one-counter machines with a single parameter.

We further show in later sections how the decidability results generalise to the reachability in simple programs of O. Ibarra *et al.* [14]—another long-standing open problem.

Our results rest in part on new developments in the theory of one-counter machines [10] and their encodings in Presburger arithmetic [9]. As no upper bounds are imposed on the value of the counter in the above publications and as their techniques make crucial use of the unboundedness of the counter, their results are, however, not directly applicable in the present setting.

8. Machines with Constant Updates

In Section 6.2 we showed that the emptiness problem for parametric timed automata with one parametric clock reduces to the emptiness problem for parametric bounded one-counter machines with all counter updates either $-1, 0$ or $+1$. For the rest of the section, fix a machine C of the latter type. We now show how to decide the emptiness problem for C.

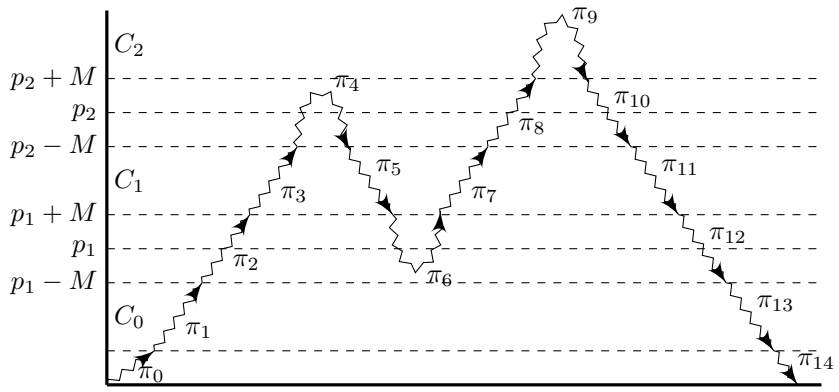


Figure 11: Factoring of a run, which starts and finishes with the counter equal to 0. The y axis shows the counter value, hypothetical values of the parameters and their neighbourhoods. Label C_i marks the interval corresponding to counter machine C_i.

To show that C halts, we have to find an assignment γ and an accepting run π in C^γ. Even without knowing γ, we show that π splits into subruns of a simple form independent of γ the existence of which is reducible to satisfiability of certain $\exists$PAD formulae.

Let γ be a parameter assignment and assume that we guessed the order of parameters, let's say, $\gamma(p_1) < \gamma(p_2) < \ldots < \gamma(p_k)$, but not the precise values of parameters. Let $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ be arbitrary configurations of C^γ such that $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ in C^γ.

We now show how to decide the existence of a run $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ in C^γ. Consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$. Note that for any constant $M \in \mathbb{N}$, the run π can be factored into subruns between successive parameters and subruns around individual parameters (see Figure 11). Formally, $\pi = \pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_l$ such that (π_0 can be possibly empty)

- Even-indexed runs: $\gamma(p_r) - M \leq \text{counter}(\pi_{2i}) \leq \gamma(p_r) + M$ for some parameter p_r,
- Odd-indexed runs: $\gamma(p_r) + M < \text{counter}(\pi_{2i+1}) < \gamma(p_{r+1}) - M$ for some consecutive parameters $\gamma(p_r) < \gamma(p_{r+1})$,
- Even- and odd-indexed runs are joined by an edge: $\text{last}(\pi_i) \to \text{first}(\pi_{i+1})$.

Now, notice that every transition in C changes the counter by at most 1. Hence, $\text{counter}(\text{first}(\pi_{2i})) = p_r + M$ or $\text{counter}(\text{first}(\pi_{2i})) = p_r - M$ for some parameter p_r. Similarly, $\text{counter}(\text{first}(\pi_{2i+1})) = p_r + M + 1$ or $\text{counter}(\text{first}(\pi_{2i+1})) = p_{r+1} - M - 1$. Thus, $\text{first}(\pi_i)$ is always of the form $\text{first}(\pi_i) = (s_i, p_{f(i)} + x)$ for some state s_i, some $|x_i| \in \{M, M+1\}$ and parameter $p_{f(i)}$. Hence, $\text{first}(\pi_i)$ is uniquely determined by the triple $(s_i, f(i), x_i)$. Similarly, $\text{last}(\pi_i)$ is uniquely determined by some triple $(t_i, g(i), y_i)$ with $|y_i| \in \{M, M+1\}$.

By minimality, π visits every configuration at most once. Hence an odd-indexed run can start in only one of $2nk$ configurations (n states, k parameters). Hence, the number of odd-indexed runs, and hence the total number of runs is $O(nk)$.

Lemma 8.1. *Let π be a shortest accepting run in C^γ. Then the factoring of π has $O(nk)$ parts.*

In Lemma 8.3, we show how to pick a small M such that we can then decide the existence of a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$.

Assumming M is fixed, to decide the existence of a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ we guess a factoring of the above form. First we show how to decide the existence of individual even- and odd-indexed runs, which we later combine to decide the existence of entire factorings.

We begin by showing how to calculate the odd-indexed runs. Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be configurations of C^γ between two successive parameters:

$$\gamma(p_i) < \text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}) < \gamma(p_{i+1}).$$

Consider a counter machine C_i, which is obtained from C by evaluating all comparisons as if the counter was between $\gamma(p_i)$ and $\gamma(p_{i+1})$. Formally, C_i is obtained from C by

- removing all transitions of the form $\geq p_j$ for $j > i$,
- removing all transitions of the form $\leq p_k$ for $k \leq i$,
- all transitions $\leq p_j$ for $j > i$ are replaced by $+0$ transitions in C_i,
- all transitions $\geq p_k$ for $k \leq i$ are replaced by $+0$ transitions in C_i,
- for $i > 0$ and $c \in \mathbb{N}$ we also remove all $\leq c$ transitions from C_i.

Note that the definition of C_i's depends only on the order of parameters in γ. Then provided the counter value stays between $\gamma(p_i)$ and $\gamma(p_{i+1})$ the runs in C_i^γ and C^γ coincide.

Recall that for a one-counter machine Z, configurations $\boldsymbol{c}, \boldsymbol{d}$ of Z and numbers $x, y \in \mathbb{N}$, we write $(\boldsymbol{c}, \boldsymbol{d}) \in Z(x, y)$ if there is a run $\pi : \boldsymbol{c} \to \boldsymbol{d}$ such that the counter stays between x and y, i.e., $x < \text{counter}(\pi) < y$. Then we have:

Lemma 8.2. *Let γ be an assignment with $\gamma(p_i) < \gamma(p_{i+1})$ for every i. Let $\boldsymbol{c}, \boldsymbol{d}$ be configurations with $\gamma(p_i) < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1})$. Then*

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff (\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$$

Proof. Immediate from the definition of C_i. □

Now, consider an arbitrary run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C_i. During the run, the counter value can become less than $\gamma(p_i)$ or greater than $\gamma(p_{i+1})$. So π does not necessarily correspond to a run in C. However, notice that C_i is a one-counter machine without parameters or $\leq x$ constraints, i.e. an ordinary one-counter machine. Thus C_i has the following property [16]: If there is a run between two configurations then there is a run between the same configurations such that the run does not deviate much from the initial and the final counter value. Formally:

Lemma 8.3 ([16], Lemma 42)**.** *Let X be a one-counter machine. There is a constant M (polynomial in $|X|$ and the magnitude of the largest constant appearing in X) such that for any configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ of X the following holds:*

- *Let $U = \min(\text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}))$ and $V = \max(\text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}))$.*
- *If $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ then there is a run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ such that $U - M \leq \text{counter}(\pi) \leq V + M$.*

So as long as $\gamma(p_1) + M < \text{counter}(\boldsymbol{c_1}), \text{counter}(\boldsymbol{c_2}) < \gamma(p_2) - M$, the runs $\boldsymbol{c_1} \to \boldsymbol{c_2}$ in C_i correspond to runs in C.

Lemma 8.4. *Let γ be an assignment with $\gamma(p_i) + M < \gamma(p_{i+1})$ for all i. Let $\boldsymbol{c}, \boldsymbol{d}$ be configurations with $\gamma(p_i) + M < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1}) - M$. Then*

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff \boldsymbol{c} \to^* \boldsymbol{d} \text{ in } C_i^\gamma.$$

Proof. Since C_i simulates C in the interval $(\gamma(p_i), \gamma(p_{i+1}))$ it is obvious that the biimplication holds if we restrict to runs inside the interval $(\gamma(p_i), \gamma(p_{i+1}))$ (Lemma 8.2):

$$(\boldsymbol{c}, \boldsymbol{d}) \in C^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff (\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$$

It thus suffices to show that

$$(\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1})) \iff \boldsymbol{c} \to^* \boldsymbol{d} \text{ in } C_i^\gamma$$

The left-to-right implication is immediate. For the converse, suppose that $\boldsymbol{c} \to^* \boldsymbol{d}$ in C_i^γ. Since, $\gamma(p_i) + M < \text{counter}(\boldsymbol{c}), \text{counter}(\boldsymbol{d}) < \gamma(p_{i+1}) - M$, by Lemma 8.3, there exists a run π in C_i^γ from $\boldsymbol{c}$ to $\boldsymbol{d}$ such that $\gamma(p_i) < \text{counter}(\pi) < \gamma(p_{i+1})$. Hence $(\boldsymbol{c}, \boldsymbol{d}) \in C_i^\gamma(\gamma(p_i), \gamma(p_{i+1}))$ as required. □

For the even-indexed runs, the reachability around individual parameters, i.e. in intervals $(\gamma(p_i) - M, \gamma(p_i) + M)$, can be precomputed. Suppose that $\gamma(p_{i-1}) < \gamma(p_i) - M < \gamma(p_i) + M < \gamma(p_{i+1})$ so that the interval $(\gamma(p_i) - M, \gamma(p_i) + M)$ does not contain $\gamma(p_{i-1})$ or $\gamma(p_{i+1})$. Let $-M < x, y < M$ and let π be a run from $(s, \gamma(p_i) + x)$ to $(t, \gamma(p_i) + y)$ such that $\gamma(p_i) - M \leq \text{counter}(\pi) \leq \gamma(p_i) + M$. Then for every component $\pi(j)$, we can write $\text{counter}(\pi(j)) = \gamma(p_i) + z_j$ for some $-M \leq z_j \leq M$. But now, the run π is valid for any specific value of $\gamma(p_i)$ as only z_j determines which transitions are enabled in C^γ.

Lemma 8.5. *Let γ, δ be parameter assignments with $\gamma(p_i) + M < \gamma(p_{i+1}), \delta(p_i) + M < \delta(p_{i+1})$ for all i. Let $s, t \in c$ be states and $-M < x, y < M$ integers. Then*

$$\begin{aligned}((s, \gamma(p_i) + x), (t, \gamma(p_i) + y)) \in C^\gamma(\gamma(p_i) - M, \gamma(p_i) + M) \quad \iff \\ ((s, \delta(p_i) + x), (t, \delta(p_i) + y)) \in C^\delta(\delta(p_i) - M, \delta(p_i) + M)\end{aligned}$$

Furthermore, it is decidable in polynomial time whether $((s, \gamma(p_i)+x), (t, \gamma(p_i)+y)) \in C^\gamma(\gamma(p_i)-M, \gamma(p_i)+M)$ for any (and all) such assignment γ.

Proof. For any run π from $(s, \gamma(p_i)+x)$ to $(t, \gamma(p_i)+y))$ in C^γ such that $\gamma(p_i)-M < \text{counter}(\pi) < \gamma(p_i)+M$ and for any index j we can write $\text{counter}(\pi(j)) = \gamma(p_i) + z_j$ for some $-M \leq z_j \leq M$. But now, the run π is valid for any specific value of $\gamma(p_i)$ as only z_j determines which transitions are enabled in C^γ.

Thus consider the graph G with vertices (s, z) where s is a state of C and $-M < z < M$. There is an edge from (s, z) to (s', z') in G if and only if there is an edge $(s, z' - z, s')$ in C, which goes from s to s' and updates the counter appropriately. Similarly, if there is an edge $(s, \leq p_i, s')$ in C then we add an edge $(s, z) \to (s', z)$ for all $z \leq 0$. Similarly for other cases.

Now, any run in C completely contained in $(\gamma(p_i) - M, \gamma(p_i) + M)$ corresponds to a path in G and vice versa. Recall, Lemma 8.3, that M is polynomial in $|C|$. Thus, $|G|$ is also polynomial in $|C|$.

Therefore, for every pair of states s and t and values z, z' in $(-M, M)$, we can calculate using a standard graph algorithm (e.g. breadth-first search), whether there is a path from $(s, \gamma(p_i) + z)$ to $(t, \gamma(p_i) + z')$. □

Combining the previous lemmas on decidability of even- and odd-indexed runs, We now prove that the existence of a run between any two configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ is definable in the existential fragment of Presburger Arithmetic with Divisibility (∃PAD definable).

Theorem 8.6. *Given states $s, t \in C$ the set $G(C, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \to^* (t, y)$ in C^γ where $\gamma(p_i) = n_i\}$ is* ∃PAD *definable.*

Proof. First, consider the case such that $n_i + M < n_{i+1}$ for every i. The variable n_i denotes the value of $\gamma(p_i)$ in the constructed ∃PAD formulae. We encode the existence of a factoring $\pi = \pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_l$ as a ∃PAD formula.

Note that $\text{first}(\pi_i)$ is always of the form $\text{first}(\pi_i) = (s_i, p_{f(i)} + x)$ for some $|x_i| \in \{M, M + 1\}$ and parameter $p_{f(i)}$. Hence, $\text{first}(\pi_i)$ is determined by the triple $(s_i, f(i), x_i)$. Similarly, $\text{last}(\pi_i)$ is determined by some triple $(t_i, g(i), y_i)$ with $|y_i| \in \{M, M + 1\}$.

Now, in Lemma 8.4 we showed that the odd-indexed runs π_{2i+1} correspond to runs in some one-counter machine $C_{h(2i+1)}$. By Lemma 5.1, the existence of a run in $C_{h(2i+1)}$ is ∃PAD expressible as:

$$\varphi_{2i+1} = \text{Reach}(C_{h(2i+1)}, s_{2i+1}, t_{2i+1})(n_{f(2i+1)} + x_{2i+1}, n_{g(2i+1)} + y_{2i+1}, n_1, \ldots, n_k)$$

where $h(2i + 1)$ denotes the appropriate one-counter machine $C_{h(2i+1)}$ the run π_{2i+1} lies in.

In Lemma 8.5, we showed that the even-indexed runs are independent of γ, can be precomputed and the reachability relation can be hardwired into the formula. By taking a conjunction of the corresponding ∃PAD formulae we obtain a single ∃PAD formula. Precisely, given states $s_1, \ldots, s_l$ and $t_1, \ldots, t_l$ and offsets $x_1, \ldots x_l$ and $y_1, \ldots y_l$ and indices $f(1), \ldots, f(l)$ and $g(1), \ldots, g(l)$ and $h(1), \ldots h(l)$, consider the formula:

$$
\begin{aligned}
G(s,t,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y},f,g,h)(x,y,\overrightarrow{n}) \;=\; & \bigwedge_i \varphi_{2i+1} \\
& \wedge\, s = s_1 \;\wedge\; n_{f(1)} + x_1 = x \wedge\; t = t_l \;\wedge\; n_{g(l)} + y_l = y \\
& \wedge\, \psi(f,g,h,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y}) \\
& \bigwedge_i n_i + M < n_{i+1}
\end{aligned}
$$

The formula asserts that there is a run from (s,x) to (t,y) with the particular factoring. The conjunct $\psi(f,g,h,\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y})$ encodes that the even-indexed subruns are valid (precomputed using Lemma 8.5) and that odd- and even-indexed runs are adjacent (directly computable) and that the values of $f(i), g(i)$ and $h(i)$ are consistent (directly computable)[6]. The last conjunct encodes the restriction $\gamma(p_i)+M < \gamma(p_{i+1})$ from Lemmas 8.4 and 8.5.

This final restriction is relaxed as follows. If the parameters are not in the increasing order $\gamma(p_i) < \gamma(p_{i+1})$ then we build appropriate formulae by relabelling the parameters so that the resulting permutation of parameters is in increasing order.

If $\gamma(p_i) \leq \gamma(p_{i+1}) < \gamma(p_i) + M$ then note that M depends only on $|C|$ and so only finitely many possibilities exist for $\gamma(p_{i+1}) - \gamma(p_i)$. Hence we replace each occurrence of p_{i+1} in C by $p_i + w$ for the appropriate $w < M$.

Now, there are only finitely many possibilities for $\overrightarrow{s},\overrightarrow{t},\overrightarrow{x},\overrightarrow{y},f,g$ and h. So all such formulae together define the reachability relation in C. Hence, the relation is $\exists$PAD definable. □

Recall that satisfiability of $\exists$PAD formulae is in NEXP [18] and for a parametric timed automaton A with one parametric clock the corresponding one-counter machine C is exponential in size in $|A|$ (Lemmas 6.3 and 6.4). Hence, the emptiness problem for A is decidable in 2NEXP time (Theorem 6.5). We have also showed a lower bound of NEXP-hard (Theorem 6.6). However, as the complexity of satisfiability of $\exists$PAD formulae is still open, only known to lie between NP-hard and NEXP, the lower bound might be optimal and further progress depends on the developments in the $\exists$PAD-satisfiability problem.

9. Parametric Timed Automata with Two Parametric Clocks

We now move to study parametric one-counter machines arising from parametric timed automata with two parametric clocks. We do not solve the problem in full as this is a long-standing open problem [2]. However, we develop new counter-machines techniques to solve the problem in case of a single parameter.

So let A be a parametric timed automaton with two parametric clocks and a single parameter p. By Remark 6.9, if the corresponding parametric bounded one-counter machine C has an accepting run then it has one where the counter is bounded by $2 \cdot \gamma(p)$.

Now, notice that C has a single parameter but may contain '$\equiv 0 \bmod c_i$' or '$+[0,p]$' transitions. We first show how to decide the emptiness problem in case C has no '$\equiv 0 \bmod c_i$' or '$+[0,p]$' transitions. Later we show how to decide the emptiness problem if both types of transitions are allowed in C.

So let C be a simple parametric bounded one-counter machine (no '$\equiv 0 \bmod c$' or '$+[0,p]$' transitions) with a single parameter p. For given $k \in \mathbb{N}$ we decide the existence of an accepting run π such that $\text{counter}(\pi) \leq k \cdot \gamma(p)$ holds.[7] Now, for any such run π and index i we can write $\text{counter}(\pi(i)) = a\gamma(p) + b$ where $a \leq k$ and $b < \gamma(p)$. Since a is bounded, we build a one-counter machine G keeping a in the state space and b in the counter. We do not enforce $b < \gamma(p)$ (or any other $\leq x$ constraint) in G. Instead, we use Lemma 8.3 on G and split π into subruns close to and far from a multiple of $\gamma(p)$. We write $\pi = \tau_0 \to \pi_1 \to \tau_1 \ldots \pi_l \to \tau_l$ such that

[6] Consistent means that $h(i)$ and $h(i+1)$ are consecutive, the indices $f(i)$ and $g(i)$ are consecutive and consistent with $h(i)$, etc.

[7] $k = 2$ in case C was obtained by reduction from a parametric timed automaton with one parameter

- for every τ_i the value counter(τ_i) mod $\gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$,
- for every π_i we have counter(π_i) mod $\gamma(p) \in (M, \gamma(p) - M)$.

For a run ρ and a subset $S \subseteq \mathbb{N}$ of natural numbers, the notation counter$(\rho) \in S$ denotes the fact that for every i we have counter$(\rho(i)) \in S$.

Then we use techniques on factoring of runs analogous to those used for one parametric clock (Section 8) to build an $\exists$PAD formula encoding the existence of an accepting run. Decidability follows from the decidability of satisfiability of $\exists$PAD formulae. In general, we have:

Lemma 9.1. *Given C with one parameter p, no '$\equiv 0 \bmod c$' and no '$+[0,p]$' transitions, $k \in \mathbb{N}$ and states $s, t \in C$ the set $G(C, s, t, k) = \{(x, y, q) \mid \exists \pi : (s, x) \to (t, y) \in C^\gamma$ such that* counter$(\pi) \leq k \cdot q$ *where $q = \gamma(p)\}$ is* $\exists$PAD *definable.*

Before proving the lemma, we first show how to decide the existence of τ_i's and π_i's. Given C and $k \in \mathbb{N}$ we introduce the one-counter machine C_k. Let S be the set of states of C. Then the states of C_k are $S \times \{0, \ldots, k-1\}$. Intuitively, if $z < \gamma(p)$, the configuration $((s, i), z)$ of C_k shall represents the configuration $(s, i \cdot \gamma(p) + z)$ of C. Machine C_k has the following transitions:

- $((s, i), \pm c, (t, i))$ if $(s, \pm c, t)$ is a transition in C,
- $((s, i), +0, (t, i \pm 1))$ if $(s, \pm p, t)$ is a transition in C,
- $((s, 0), +0, (t, 0))$ if $(s, \leq p, t)$ is a transition in C,
- $((s, i), +0, (t, i))$ for $i \geq 1$ if $(s, \geq p, t)$ is a transition in C.

Intuitively, the "submachine" $S \times \{i\}$ should handle C when the counter of C lies in $[i\gamma(p), (i+1)\gamma(p)]$. Now, the simulation is not perfect, as the counter of G can be larger than $\gamma(p)$ thereby incorrectly enabling/disabling various transitions. However, note that G has no parameters or $\leq p$ transitions, i.e it is an ordinary one-counter machine. Hence, by Lemma 8.3, there is a constant $M \in \mathbb{N}$ such that we can assume that runs deviate by at most M from the initial and the final counter values.

Lemma 9.2. *Let γ be an assignments, s_1, s_2 states of C and $0 \leq a_1, a_2 < k$ and $M < b_1, b_2 < \gamma(p) - M$ be natural numbers. Then the following are equivalent.*

- *There is a run π from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$ in C^γ such that* counter$(\pi) \leq k\gamma(p)$ *and* counter$(\pi) \bmod \gamma(p) \in (M, \gamma(p) - M)$.
- *There is a run π' from $((s_1, a_1), b_1)$ to $((s_2, a_2), b_2)$ in C_k^γ.*

Proof. Analogous to lemma 8.4. Suppose that a run π exists. Then let f be a function that sends the configuration $(s, a\gamma(p)+b)$ of C^γ where $0 \leq b < \gamma(p)$ to the configuration $((s, a), b)$ of C_k^γ. By the construction of C_k and restrictions on π, the path $f(\pi)$ obtained by applying f to every component of π is a valid path in C_k^γ from $((s_1, a_1), b_1)$ to $((s_2, a_2), b_2)$.

Conversely, suppose that there is a run π'. By Lemma 8.3, we can assume that $0 <$ counter$(\pi') < \gamma(p)$. So let g be a function that sends the configuration $((s, a), b)$ of C_k^γ to the configuration $(s, a\gamma(p) + b)$ of C^γ. By the construction, the path $g(\pi')$ obtained by applying g to every component of π' is a valid path in C^γ from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$. □

The above lemma shows how to handle π_i runs. For the τ_i runs, recall that for every τ_i we have counter(τ_i) mod $\gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. As in Lemma 8.5, the existence of runs τ_i is independent of $\gamma(p)$ and can be precomputed. Consider the graph with vertices $\{0, \ldots, k\} \times \{-M, \ldots, M\}$. Each vertex (i, j) corresponds to counter value $i \cdot \gamma(p) + j$ in C^γ. The edges mimic the transitions in C_k. Now, the graph is independent of $\gamma(p)$ and so we can calculate reachability between any pair of vertices.

Lemma 9.3. *Let γ be an assignment. Given $k \in \mathbb{N}$ and states s_1, s_2 of C and numbers $0 \leq a_1, a_2 \leq k$ and $b_1, b_2 \in [0, \ldots, M] \cup [\gamma(p) - M, \ldots, M)$ it is decidable in polynomial time whether there is a path π from $(s_1, a_1\gamma(p) + b_1)$ to $(s_2, a_2\gamma(p) + b_2)$ in C^γ such that* counter$(\pi) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. *Moreover, the existence of such a path is independent of $\gamma(p)$.*

We now have all the pieces necessary to prove Lemma 9.1.

Proof. (of Lemma 9.1)

The proof is analogous to the proof of Theorem 8.6. Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be two configurations of C^γ. Consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C^γ. Then π can be split into subruns close to and far from a multiple of $\gamma(p)$. We write $\pi = \tau_0 \to \pi_1 \to \tau_1 \dots \pi_l \to \tau_l$ such that for every τ_i the value $\text{counter}(\tau_i) \bmod \gamma(p) \in [0, \dots, M] \cup [\gamma(p) - M, \gamma(p))$. And for every π_i we have $\text{counter}(\pi_i) \bmod \gamma(p) \in (M, \gamma(p) - M)$.

Notice that the configuration $\text{first}(\tau_i)$ is uniquely determined by the state of C, $\lfloor \text{counter}(\text{first}(\tau_i))/\gamma(p) \rfloor$ and $\text{counter}(\text{first}(\tau_i)) \bmod \gamma(p)$. Now, C has only finitely many states, $\lfloor \text{counter}(\text{first}(\tau_i))/\gamma(p) \rfloor \leq k$ and $\text{counter}(\text{first}(\tau_i)) \bmod \gamma(p)$ can have only one of $2M + 1$ values. Recall that π is a shortest run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$. In particular, π visits each configuration of C^γ at most once. Hence there are only $O(nkM)$ different initial configurations for τ_i's and hence $l = O(nkM)$.

Now, by Lemma 9.2, the existence of π_i can be witnessed by C_k. Further, by Lemma 9.3, the existence of runs τ_i is independent of $\gamma(p)$ and can be precomputed. The runs in C_k are $\exists$PAD expressible (Lemma 5.1). By taking a conjunction of the corresponding $\exists$PAD formulae we obtain a single $\exists$PAD formula

$$\left(\bigwedge_i \text{Reach}(C_k, \text{first}(\pi_i), \text{last}(\pi_i))(\text{counter}(\text{first}(\pi_i)), \text{counter}(\text{last}(\pi_i)), p) \right) \wedge \psi$$

defining the reachability relation for a particular factoring where (as in Theorem 8.6) the formula ψ encodes that τ_i's are valid (directly computable) and that τ_i and π_i can be connected by an edge (directly computable).

Since there are only finitely many initial and final states of π_i's and τ_i's, which uniquely determine the factoring, by taking the set of all such $\exists$PAD formulae, we conclude that the reachability relation is $\exists$PAD definable. □

9.1. Elimination of '$\equiv 0 \bmod c$' Transitions

Next, we show how to handle '$\equiv 0 \bmod c$' transitions. Let C be a parametric bounded one-counter machine with '$\equiv 0 \bmod c$' transitions. We now show how to eliminate '$\equiv 0 \bmod c$' transitions from C.

Let $K = \{c_1, \dots, c_r\}$ be the set of all constants appearing as '$\equiv 0 \bmod c_i$' in C. Intuitively, we modify C to store in its state space the counter modulo each c_i. However, knowledge of $p \bmod c_i$ for each i is necessary for that.

Given $D = (d_1, \dots, d_r)$, let C_D be the parametric bounded one-counter machine which is obtained from C and which tracks the counter modulo each c_i assuming that $p \equiv d_i \bmod c_i$. Formally, the states of C_D are $S \times \mathbb{Z}_{c_1} \times \dots \times \mathbb{Z}_{c_r}$ where S are the states of C and $\mathbb{Z}_{c_i}$ denotes the ring of integers modulo c_i. Let $(v_1, \dots, v_r) \in \mathbb{Z}_{c_1} \times \dots \times \mathbb{Z}_{c_r}$. Then C_D contains the following transitions:

- $((q, v_1, \dots, v_r), \pm c, (q', v_1 \pm c, \dots, v_r \pm c)$ if $(q, \pm c, q')$ is a transition in C,
- $((q, v_1, \dots, v_r), \pm p, (q', v_1 \pm d_1, \dots, v_r \pm d_r)$ if $(q, \pm p, q')$ is a transition in C,
- $((q, v_1, \dots, v_r), +0, (q', v_1, \dots, v_r))$ if $v_i = 0$ and $(q, \equiv 0 \bmod c_i, q')$ is a transition in C,
- $((q, v_1, \dots, v_r), G, (q', v_1, \dots, v_r))$ if (q, G, q') is a transition in C and G is a guard (comparison).

Notice that there are no '$\equiv 0 \bmod c$' transitions in C_D. By construction, runs in C_D^γ are equivalent to runs C^γ provided $d_i \equiv \gamma(p) \bmod c_i$. That is:

Lemma 9.4. *Let C be a parametric bounded one-counter machine with a single parameter p, let γ be an assignment such that $\gamma(p) = d_i \bmod c_i$ for each i. Let $(s, x), (t, y)$ be configurations of C. Then $(s, x) \to^* (t, y)$ in C^γ if and only if $((s, x \bmod c_1, \dots, x \bmod c_r), x) \to^* ((t, y \bmod c_1, \dots, y \bmod c_r), y)$ in C_D^γ.*

Proof. Let π be a run $\pi : (s, x) \to (t, y)$. Define $f : \mathbb{N} \to Z$ by $f(v) = (v \bmod c_1, \dots, v \bmod c_r)$. Let τ be the run obtained from π by sending each configuration (s_i, v_i) to $(s_i \times f(v_i), v_i)$. By construction, each transitions $(s_i \times f(v_i), v_i) \to (s_{i+1} \times f(v_{i+1}), v_{i+1})$ is valid in C_D^γ.

For the converse, for configuration $\boldsymbol{c} = ((s_i, v_1, \dots, v_r), w_i)$ let $g(c)$ be the function that gives $g(c) = (s_i, w_i)$. Let $\pi : ((s, x \bmod c_1, \dots, x \bmod c_r), x) \to ((t, y \bmod c_1, \dots, y \bmod c_r), y)$ be a run in C_D^γ. Let

τ be the run obtained by applying g to π. By construction, each transitions $(s_i, w_i) \to (s_{i+1}, w_{i+1})$ is valid in C_D^γ. □

Using the above lemma together with the result (Theorem 9.1) for machines without '$\equiv 0 \mod c$' transitions we obtain:

Theorem 9.5. *Let C be parametric bounded one-counter machine with '$\equiv 0 \mod c_i$' transitions and a single parameter p. Given $k \in \mathbb{N}$ and states s and t, the set $H(C, s, t, k) = \{(x, y, q) \mid \exists \pi : (s, x) \to (t, y) \in C^\gamma$ such that $\text{counter}(\pi) \leq k \cdot q$ where $q = \gamma(p)\}$ is $\exists$PAD definable.*

Proof. For a fixed $c \in \mathbb{N}$ the $\exists$PAD formula $\varphi_c(x, y) = (\exists q \,.\, c \cdot q + y = x) \wedge y < c$ asserts that $x \equiv y \mod c$.

Thus, given $D \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$, we construct the formula $H'(A, s, t, k, D)(x, y, p)$ asserting that D is consistent with p and that $(s, x) \to^* (t, y)$ in C_D:

$$H'(C, s, t, k, D)(x, y, p) = \exists \overrightarrow{v}(r), \overrightarrow{w}(r) \quad . \quad G(C_D, s \times v, t \times w, k)(x, y, p) \wedge \bigwedge_i [(x \equiv v(i) \mod c_i) \wedge (y \equiv w(i) \mod c_i) \wedge (p \equiv D(i) \mod c_i)]$$

Since $\exists$PAD definable sets are closed under finite union and there are only finitely many possible D's, we get the desired result. □

9.2. Elimination of '$+[0, p]$' Transitions

In this subsection suppose that the parametric bounded one-counter machine C contains '$\equiv 0 \mod c$' as well as '$+[0, p]$' transitions. We show that the reachability relation of C is $\exists$PAD definable also in this case.

Let $K = \{c_1, \ldots, c_k\}$ be the set of all constants appearing in '$\equiv 0 \mod c_i$' transitions in C and let $R = \text{lcm}(K)$ be their least common multiple. Suppose that C^γ is nonempty and take π to be a shortest accepting run in C^γ. We will show that all but a bounded number (depending only on C) of traversals of '$+[0, p]$' edges increment the counter by at most R or by at least $\gamma(p) - R$.

So suppose that at least two '$+[0, p]$' transitions appear along π. Then, we can write π as $\pi = \tau_1 \xrightarrow{e_1} \tau_2 \xrightarrow{e_2} \tau_3$ where e_1 and e_2 are '$+[0, p]$' transitions.

Denote the counter updates at e_1 and e_2 by f_1 and f_2, respectively. If $R < f_1, f_2 < \gamma(p) - R$ then consider the run τ_2' which is obtained from τ_2 by incrementing the counter by R. That is, for every index i, we have $\text{state}(\tau_2(i)) = \text{state}(\tau_2'(i))$ and $\text{counter}(\tau_2'(i)) = \text{counter}(\tau_2(i)) + R$. This corresponds to incrementing the counter in e_1, e_2 by $f_1 + R, f_2 - R$ respectively.

Now, τ_2' might not be a valid run. In particular, three things can potentially occur.

- There is a transition $(q, \text{'} \equiv 0 \mod c', q')$ from $\tau_2'(i)$ to $\tau_2'(i+1)$ for some i and $\text{counter}(\tau_2'(i)) \not\equiv 0 \mod c$. However, as $\tau_2(i) \equiv 0 \mod c$ by assumption on π and $c | R$, we have $c | \tau_2(i) + R = \tau_2'(i)$ and so this situation cannot happen.
- There is a transition $(q, \leq p, q')$ from $\tau_2'(i)$ to $\tau_2'(i+1)$ for some index i and $\text{counter}(\tau_2'(i)) > \gamma(p)$. Hence, $\text{counter}(\tau_2(i)) > \gamma(p) - R$. By assumption, we also have $\text{counter}(\tau_2(i)) \leq \gamma(p)$. Therefore, $\gamma(p) - R < \text{counter}(\tau_2(i)) \leq \gamma(p)$. But by minimality, π visits every configuration at most once and so such a situation can occur at most nR times—R times per each of n states. Otherwise, we would obtain a shorter run by cutting out the subrun between two occurrences of configuration $\tau_2(i)$.
- there exists index i such that $\text{counter}(\tau_2'(i)) > k \cdot \gamma(p)$. But then we have $k \cdot \gamma(p) - R < \text{counter}(\tau_2(i)) \leq k \cdot \gamma(p)$. By minimality, π visits every configuration at most once and so such a situation can occur at most nR times—R times per each of n states.

We use the above observations to repeatedly increment some transition $e_1 = +[0, p]$ and simultaneously decrement some other transition $e_2 = +[0, p]$ by R; eventually stopping when no such pair of transitions exists.

Formally, let e_1 be the first '$+[0, p]$' transition such that there is a transition $e_2 = +[0, p]$ so that e_1, e_2 satisfy condition (ii) or (iii) above. Then $e_i 1$ can be incremented by R and e_2 can be decremented by R.

So let π' be the run $\pi' = \tau_1 \xrightarrow{e_1} \tau_2' \xrightarrow{e_2} \tau_3$. Replace π by π' and repeat the process creating the runs $\pi', \pi'', \pi''', \ldots$. Note that counter($\pi'$) is lexicographically larger than counter(π). As π is of finite length and all transitions can be incremented by at most $|\pi|\gamma(p)$, the procedure eventually terminates for some $\pi^{(k)}$. Let $\{f_1, \ldots, f_v\}$ be the set of increments caused by '$+[0,p]$' transitions along $\pi^{(k)}$. By above, each f_i is either at most R, at least $\gamma(p) - R$ or one of at most nR different values. Thus, we have shown:

Lemma 9.6. *Let C be a parametric bounded one-counter machine with a single parameter p and $\gamma : \{p\} \to \mathbb{N}$ be an assignment. If there is an accepting (bounded) run in C^γ then there is an accepting (bounded) run π such that at most nR counter increments by '$+[0,p]$' transitions that are not in the set $\{+0, +1, \ldots, +R, +p-R, \ldots, +p\}$. That is,*

$$|\{i \mid \pi(i) = +[0,p] \text{ and } R < \text{counter}(\pi(i+1)) - \text{counter}(\pi(i)) < \gamma(p) - R\}| \leq 2nR.$$

Proof. The case of runs bounded by a multiple of $\gamma(p)$ is covered in the paragraphs above.

If a run is not bounded by a multiple of $\gamma(p)$, then note that the third condition above does not occur. Hence, the cardinality of the set under consideration is at most nR. □

We now show that the existence of such a factoring can be specified by a ∃PAD formula.

Theorem 9.7. *Given C with '$\equiv 0 \bmod c$' and '$+[0,p]$' transitions and a single parameter p. Let $s, t \in C$ be states. Then the set $I(C,s,t,k) = \{(x,y,q) \in \mathbb{N}^3 \mid \exists \pi : (s,x) \to^* (t,y)$ in C^γ such that counter(π) $\leq k \cdot \gamma(p)$ where $\gamma(p) = q\}$ is ∃PAD definable.*

Proof. Let Z be the parametric bounded one-counter machine obtained from C by replacing each '$+[0,p]$' transition by $2R+2$ transitions: $+0, +1, \ldots, +R, +p-R, \ldots, +p$. Then $\pi^{(k)}$ (as defined in a paragraph above) can be factored as $\pi^{(k)} = \pi_0 \to \pi_1 \to \cdots \to \pi_v$ where each π_i is a run in Z^γ and there is a '$+[0,p]$' transition between π_i and π_{i+1}.

Given $v \leq 2nR$ and states $s = s_1, s_2, \ldots, s_v, t_1, \ldots, t_{v-1}, t_v = t$ the formula

$$I'(C,s,t,v,k,\overrightarrow{s},\overrightarrow{t})(x,y,p) = \exists \overrightarrow{x}, \overrightarrow{y} \; . \quad \bigwedge_{i=1\ldots v} I(Z,s_i,t_i,k)(x_i,y_i,p) \;\wedge\; x_1 = x \;\wedge\; y_v = y$$
$$\bigwedge_{i=1\ldots v-1} (0 \leq y_{i+1} - x_i \leq p \;\wedge\; E(t_i, \text{`}+[0,p]', s_{i+1})$$

asserts that such a factoring of length v exists where first(π_i) $= (s_i, x_i)$ and last(π_i) $= (t_i, y_i)$. The predicate $E(q, \text{`}+[0,p]', q')$ encodes that there is a '$+[0,p]$' transition between q and q' in C.

Since $v \leq 2nR$ there are only finitely many possibilities for $\overrightarrow{s}$ and $\overrightarrow{t}$. As ∃PAD sets are closed under finite union, the result follows. □

As an immediate corollary (using Remark 6.9) we get the following result on parametric timed automata:

Theorem 9.8. *The emptiness problem is decidable for parametric timed automata with two parametric clocks and a single parameter.*

10. Parametric Bounded One-Counter Machines with One Parameter

In the previous section we showed how to decide the emptiness problem for parametric timed automata with two parametric clocks with a single parameter. The decidability proof works by reduction to parametric bounded one-counter machines with a single parameter p and accepting runs with counter(π) $\leq 2\gamma(p)$. We now relax the restriction on counter(π) $\leq 2\gamma(p)$. This simple modification leads to a substantially complicated proof based around the analysis of subruns exceeding $2\gamma(p)$. See Figure 12 showing the reductions between the automata, one-counter machines classes and Presburger arithmetic with divisibility.

Let C be a parametric bounded one-counter machine with a single parameter. We show in this section how to decide the emptiness problem for C. Note that by Theorem 6.7, machine C has the emptiness problem equivalent to some parametric timed automaton with two parametric clocks with *two* parameters.

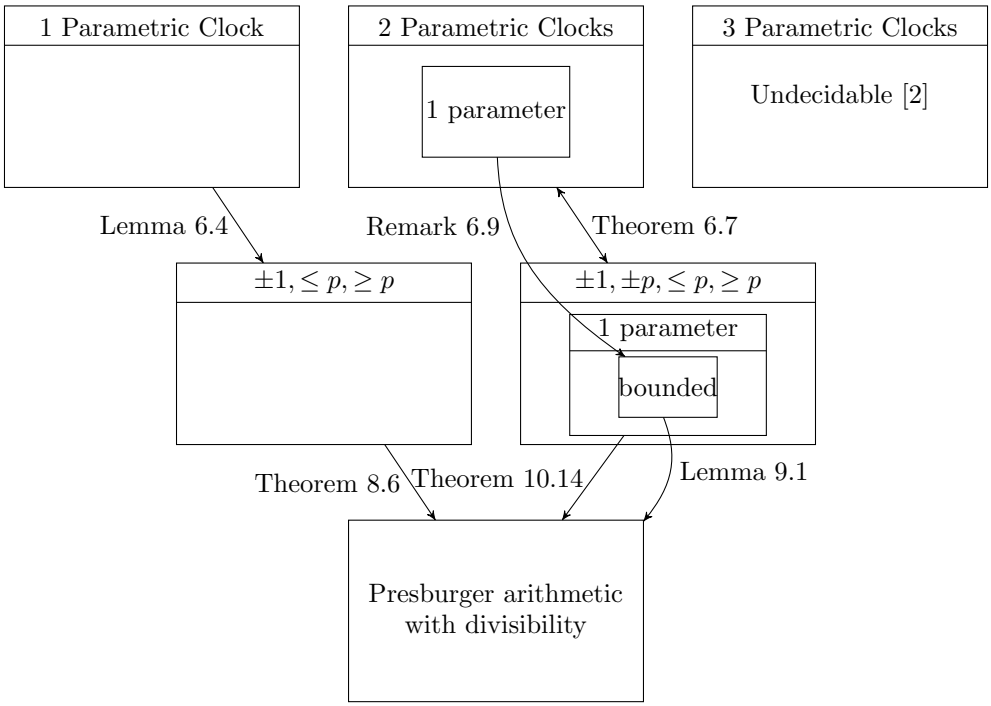


Figure 12: Diagram of reductions from Parametric Timed Automata to Parametric One-Counter Machines and Presburger Arithmetic with Divisibility

	simple	'$\equiv 0 \bmod c$'	'$\equiv 0 \bmod c$', '$+[0,p]$'
bounded runs	Theorem 9.1	Theorem 9.5	Theorem 9.7
unbounded runs	Theorem 10.14	Theorem 10.16	Theorem 10.17

Table 1: Parametric Bounded One-Counter Machines, One Parameter Case

However, the decidability of the emptiness problem for the latter class is open in general. Thus, solving decidability in this special class of one-counter machines is a necessary step towards resolving the general case of the emptiness problem for parametric timed automata.

We begin by considering the situation when C has neither '$\equiv 0 \bmod c$' nor '$+[0,p]$' transitions. The results for the various cases of the single parameter problem are summarised in Table 1.

Firstly, to handle the corner cases when the value of $\gamma(p)$ is too small, the decision procedure begins by instantiating $\gamma(p)$ at each value in $[0, \ldots, c_{max}]$ and checking whether the final state is reachable in the resulting nonparametric one-counter automaton. The value c_{max} denotes the largest constant used in C. Since reachability is decidable in nonparametric bounded one-ounter automata, determining whether there is an accepting run in C^γ for some $\gamma(p) \leq c_{max}$ is decidable. Thus, for the remaining sections we assume that $\gamma(p)$ is greater than the largest constant c_{max} occuring in C.

In general, the decidability of the emptiness problem for C is quite involved and depends on the theory of ordinary parametric one-counter machines [9] (summarised in Section 10.3). We begin by giving a quick overview.

10.1. Proof Outline For Simple Parametric Bounded One-Counter Machines

We first show the result for simple parametric bounded one-counter machines (no '$\equiv 0 \bmod c$' or '$+[0,p]$' transitions). To show that such a machine C halts, we have to find an assignment γ and an accepting run

π in C^γ. So suppose that there is some γ, consider any accepting run π in C^γ and factor π as follows

$$\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t \tag{1}$$

where $\text{counter}(\tau_i) \leq 2\gamma(p) < \text{counter}(\pi_i)$ for every i. Notice that since $\text{counter}(\pi_i) \geq 2\gamma(p)$, the runs π_i's do not use any $\leq p$ transitions. Intuitively, such runs correspond to runs in some one-counter machine. Using theory of one-counter machines developed in [9, 10], each individual π_i can be factored in a special way. Building upon that theory, we show in Section 10.2 that there is $K \in \mathbb{N}$ depending only on C (and not on γ) such that all but a bounded number of π_i's have $\text{counter}(\pi) \leq K \cdot \gamma(p)$.

Rewriting (1), we then have that there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ and $t \leq L$ and $\text{counter}(\tau_i) \leq K \cdot \gamma(p) < \text{counter}(\pi_i)$ for every i.

In Theorem 10.14 we use results from [9, 10] and Theorem 9.1 to show that both π_i's and τ_i's are $\exists$PAD definable, respectively. Hence, $\exists$PAD definability of reachability in C follows.

For the rest of the section, fix a simple parametric bounded one-counter machine C with a single parameter p. Denote the number of states of C by n. We show how to obtain a factoring of runs in C^γ.

10.2. Properties of Parametric Bounded One-Counter Machines

Let γ be an assignment and π a run in C^γ. As in (1), write π as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ such that $\text{counter}(\tau_i) \leq 2\gamma(p) < \text{counter}(\pi_i)$ for every i. The goal of this section is to show that irrespective of γ, all but a small number of π_i's are bounded by $K \cdot \gamma(p)$ for some constant $K \in \mathbb{N}$ where the value of K as well as the number of subruns π_i's violating the $K \cdot \gamma(p)$ upper bound depends only on C (and is independent of γ).

Notice that since $\text{counter}(\pi_i) \geq 2\gamma(p)$, the runs π_i's do not use any $\leq p$ transitions. Hence, we can think of them as runs in a one-counter machine. The machine $C_{>p}$ is obtained from C by removing all $\leq p, \leq c$ transitions and leaving all other transitions and states unchanged. Then $C_{>p}$ is a parametric one-counter machine without upper bounds. Moreover, $C_{>p}$ simulates runs of C when the counter stays above $\gamma(p)$. Formally,

Lemma 10.1. *Let $s, t \in C$ be states of C and $0 < x, y \in \mathbb{N}$ be positive natural numbers. Then there is a run $\pi : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in C^γ such that $\text{counter}(\pi) > \gamma(p)$ if and only if there is a run $\pi' : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in $C^\gamma_{>p}$.*

Proof. Consider $\pi : (s, \gamma(p) + x) \to (t, \gamma(p) + y)$ in C^γ such that $\text{counter}(\pi) > \gamma(p)$. Then π does not use any $\leq p$ transitions. Hence, π is a run in $C^\gamma_{>p}$. The converse is symmetric. □

We now turn to study runs in parametric one-counter machines. In the following section we study properties of runs in one-counter machines that will be useful to bound subruns π_i's.

10.3. Properties of Runs in One-Counter Machines

Let Z be a one-counter machine (with no upper bounds and no parameters). The reachability problem for such machines was shown decidable in [10, 9] and a result of [10, 9] also characterised the reachability between any two configurations. We now outline the results obtained in those publications.

Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be two configurations of Z. We aim to show that if there is a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ then there is one of a special form. So suppose that there is a run π from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$.

Further, suppose that there is loop with positive net effect followed by a loop with negative net effect in π. Let α be the first loop in π with a positive net effect and let β be the last loop in π with negative net effect: $\text{effect}(\alpha) > 0 > \text{effect}(\beta)$. Then π can be written as:

$$\pi = \pi_1 \to \alpha \to \tau \to \beta \to \pi_3.$$

Notice that for any $k \in \mathbb{N}$, we have $k|\,\text{effect}(\beta)|\,\text{effect}(\alpha) + k\,\text{effect}(\alpha)\,\text{effect}(\beta) = 0$. Therefore, the run $\pi_k = \pi_1 \to \alpha^{(1+k|\,\text{effect}(\beta)|)} \to \tau \to \beta^{(1+k|\,\text{effect}(\alpha)|)} \to \pi_3$ obtained from π by taking the the loop α exactly

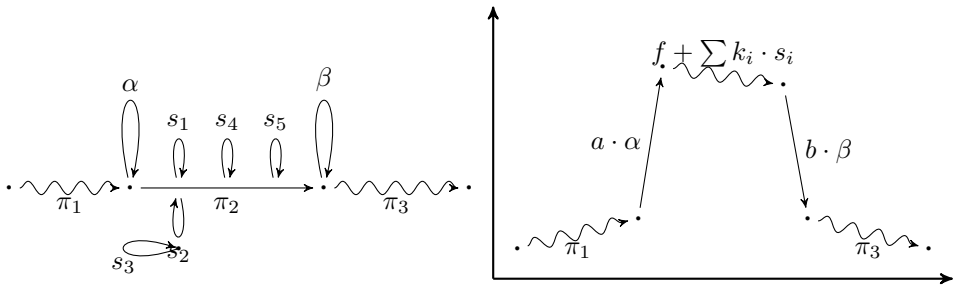


Figure 13: Factoring of a run as $\pi_1 \cdot \pi_2 \cdot \pi_3$ is shown on the left. The right figure shows how the counter value changes during the run. The loops α and β are positive and negative respectively. Thus, they can be used to pump up the counter arbitrarily high and then bring it back. Hence, the order of traversals of loops $s_1, \ldots, s_5$ does not matter.

$1 + k|\operatorname{effect}(\beta)|$ times and by taking the loop β exactly $1 + k \operatorname{effect}(\alpha)$ times starts and finishes in the same configuration as π does. Moreover, as α is a positive loop, the counter along the run π_k never becomes negative and so π_k is a valid path in Z for every $k > 0$.

Therefore, the counter value in τ can be made arbitrarily large by pumping the counter arbitrarily high using α and then bringing the counter back using β. In particular, we can ignore in τ the order in which individual loops are visited as we can assume that the counter is large enough so that no transition makes counter negative. Thus, the precise order of transitions in π is irrelevant as long as it gives one connected path. Such path are described as follows. (See also Figure 13)

Definition 10.2. *For simple loops s_i, path f and numbers k_i we say that path π is of the form $f + \sum_i k_i s_i$ if the expression describes how many times each transition of Z is visited by π. Formally, for each transition e in Z, we have* $\operatorname{count}(\pi, e) = \operatorname{count}(f, e) + \sum_i k_i \operatorname{count}(s_i, e)$ *where the expression* $\operatorname{count}(X, e)$ *denotes the number of occurrences of a transition e in the set X.*

Further, by removing nested negative loops from α, we can assume that α is a simple loop. Similarly, by removing nested positive loops from β, we can assume that β is a simple loop. Formally, the following result appeared in [10, 9].

Lemma 10.3 ([10, 9])**.** *Let Z be a one-counter machine (without upper bounds or parameters). For configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ of Z, if there is a run $\tau : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in Z then there is a run π from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ such that $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ and*

- $\min \operatorname{counter}(\pi) \geq \min \operatorname{counter}(\tau)$,
- *There are no positive simple loops in π_1,*
- *There are no negative simple loops in π_3,*
- *$\pi_2 = \alpha^a \to \sigma \to \beta^b$ where α and β are simple loops such that* $\operatorname{effect}(\beta) < 0 < \operatorname{effect}(\alpha)$ *and σ is of the form $(f + \sum k_i s_i)$ where f is a simple path, s_i's are simple loops, $a, b, k_i \in \mathbb{N}$ and the set of edges $f \cup \{s_i\}_i$ is connected.*

We denote by $\operatorname{type}(\pi)$ *the tuple* $(\operatorname{support}(\pi_1), \alpha, \beta, f, \{s_i\}, \operatorname{support}(\pi_3))$*. Runs of this form are called* factored*. If $\pi_2 \neq \epsilon$ then we say that π is* pumpable*.*

Note that there are only finitely many types. Moreover, $\operatorname{div}(\tau) \leq \operatorname{div}(f) + \sum_i k_i \operatorname{div}(s_i) \leq \operatorname{div}(f) + n \sum_i k_i M$ where M is the largest constant appearing in Z.

Now, in the middle segment π_2 only the number of traversal of loops matters as, by pumping α and β if necessary, we can always assume that the counter value is large enough. Therefore, the above Lemma has a converse stating that once the numbers of traversals are specified a run with the corresponding effect always exists[8]:

[8]The run is a collection of Eulerian traversals of the graph spanned by f and $\{s_i\}_i$ with the appropriate multiplicities.

Lemma 10.4 ([9]). *Let f be a simple path, s_i be simple loops such that $f \cup \{s_i\}_i$ is connected. Let π_1, π_3 be runs such that $\mathrm{last}(\pi_1) = \mathrm{first}(f)$ and $\mathrm{last}(f) = \mathrm{first}(\pi_3)$ and let $a, b, k_i > 0$ be natural numbers. Then there is a run π in Z of the form $\pi_1 \to \alpha^{a'} \to f + \sum_i k_i s_i \to \beta^{b'} \to \pi_3$ such that $\mathrm{effect}(\pi) = \mathrm{effect}(\pi_1) + a\,\mathrm{effect}(\alpha) + \mathrm{effect}(f) + \sum_i k_i\,\mathrm{effect}(s_i) + b\,\mathrm{effect}(\beta) + \mathrm{effect}(\pi_3)$.*

In particular, by modifying number of traversals of loops s_i in a factored run it is always possible to create another factored run with the corresponding change in effect. Formally,

Lemma 10.5. *Let $\pi = \pi_1 \to \alpha^a \to (f + \sum k_i s_i) \to \beta^b \to \pi_3$ be a factored run. Let $u, v \in \{a, b, k_1, k_2, \ldots\}$ and $\Delta_u, \Delta_v \in \mathbb{N}$ then if $u+\Delta_u, v+\Delta_v > 0$ then there is a run π', denoted as $\pi' = \pi(u \leftarrow u+\Delta_u, v \leftarrow v+\Delta_v)$, such that*[9]

- $\mathrm{effect}(\pi') = \mathrm{effect}(\pi) + \Delta_u\,\mathrm{effect}(\sigma_u) + \Delta_v\,\mathrm{effect}(\sigma_v)$,
- $\mathrm{counter}(\pi') \geq \min(\mathrm{counter}(\pi), \mathrm{counter}(\pi) + \Delta_u\,\mathrm{effect}(\sigma_u) + \Delta_v\,\mathrm{effect}(\sigma_v))$

where σ_u, σ_v are the loops associated with u and v, respectively.

Proof. Let π' be the path as in the statement. Such a path exists by Lemma 10.4. Write π' as $\pi' = \rho_1\rho_2\rho_3$ where

- ρ_1 corresponds to the prefix $\pi_1 \to \alpha^{a'}$,
- ρ_2 corresponds to the middle segment $f + \sum k_i' \cdot s_i'$,
- ρ_3 corresponds to the suffix $\beta^{b'} \to \pi_3$.

Since up to possibly different number of traversals of the positive loop α the path ρ_1 agrees with a prefix of π, we have $\mathrm{counter}(\rho_1) \geq \min \mathrm{counter}(\pi)$.

Further, by modifying a' and b' if necessary, we can assume that $\min(\mathrm{counter}(\rho_2))$ is sufficiently large and hence greater than $\min \mathrm{counter}(\pi)$.

Finally, note that ρ_3 corresponds to a suffix of π where the counter is incremented by $\Delta_u\,\mathrm{effect}(\sigma_u) + \Delta_v\,\mathrm{effect}(\sigma_v)$. Hence, $\mathrm{counter}(\rho_3) \geq (\min \mathrm{counter}(\pi)) + \Delta_u\,\mathrm{effect}(\sigma_u) + \Delta_v\,\mathrm{effect}(\sigma_v)$.

By taking the minimum over the three cases above, the result follows. □

Going back to the factoring (1) of an accepting run π in the parametric bounded one-counter machine C^γ, we deduce from Lemmas 10.3 and 10.1 that every π_i can be factored.

Lemma 10.6. *Let c_1, c_2 be two configurations in C^γ. There is a run $\pi : c_1 \to c_2$ such that $\mathrm{counter}(\pi) > 2\gamma(p)$ if and only if there is a factored run $\pi' : c_1 \to c_2$ such that $\mathrm{counter}(\pi') > 2\gamma(p)$.*

Proof. The right-to-left implication is immediate. For the left-to-right implication, consider the one-counter machine $C_{>p}$. Now, π is a run in C^γ such that $\mathrm{counter}(\gamma) > \gamma(p)$. Hence, π does not traverse any $\leq p$ transitions in C. Hence π is a run in $C^\gamma_{>p}$. Applying Lemma 10.3 to π and $C_{>p}$ gives the desired result. □

So we assume that every π_i in the factoring (1) is factored. In the following lemmas we study factored runs of different types in order to bound the factoring (1).

10.4. Properties of Factored Runs

In this section we prove tighter bounds on the structure of factored runs. Recall that in Lemma 10.3 we have associated a type with every run. In the following sections, we partition all factored runs according to their types into three classes (one class for nonpumpable runs and two classes for pumpable runs). We first show that all nonpumpable as well as one class of pumpable runs is essentially bounded. Then we show that the remaining pumpable runs can appear only a bounded number of times.

[9] The statement naturally generalise to modifying the number of traversals of any number of loops. However, in all the results in this manuscript we modify exactly two loops and therefore present the result only in this special case.

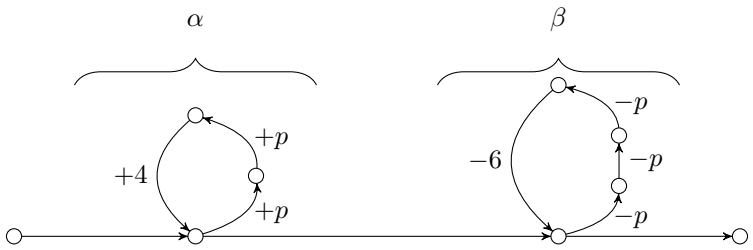


Figure 14: Counter machine with only linear runs. The automaton has two loops. A positive loop α with effect(α) $= 2p + 4$ and a negative loop β with effect(β) $= -3p - 6$. Notice that $\alpha \times \beta = 0$ and the common denominator of the loops is $p + 2$.

10.5. Nonpumpable Runs

First, consider nonpumpable runs in C^γ. The following lemma says that nonpumpable runs are bounded.

Lemma 10.7. *Let $\pi = \pi_1\pi_2\pi_3$ be a nonpumpable run in C^γ. Then*

$$\text{counter}(\pi) \leq n\gamma(p) + \max(\text{counter}(\text{first}(\pi)), \text{counter}(\text{last}(\pi))).$$

Proof. A simple path in C is always of length at most n and a transition increases the counter by at most $\gamma(p)$. Thus, along any simple path, the counter increases by at most $n\gamma(p)$. Now, π_1 has no positive loops. So $\text{div}(\pi_1) \leq n\gamma(p)$ and hence, $\text{counter}(\pi_1) \leq \text{counter}(\text{first}(\pi_1)) + n\gamma(p)$.

Similarly, starting from last(π) and considering the reverse of the path π_3 we obtain $\text{counter}(\pi_3) \leq \text{counter}(\text{last}(\pi_3)) + n\gamma(p)$. Combining the two gives the result. □

10.6. Pumpable Runs

Next, consider pumpable runs. For such runs, the cross product between positive and negative loops occurring in π_2 will play a crucial role.

For a run π in C^γ, by counting the number of $\pm p$ and the net effect of $\pm c$ transitions in π there is a natural way of writing $\textbf{effect}\,(\boldsymbol{\pi}) = \boldsymbol{a\gamma(p) + b}$ for $a, b \in \mathbb{Z}$ (see e.g., Figure 14 and Figure 15).

Let σ, τ be two runs and write $\text{effect}(\sigma) = a\gamma(p) + b$ and $\text{effect}(\tau) = c\gamma(p) + d$. By the *cross product* of σ, τ we mean $\sigma \times \tau = ad - bc$.

Definition 10.8. *Let T be a type of a factored run and let P and N be the set of positive and negative loops in T, respectively. We say that T (or a run π of type T) is* linear *if for every $\rho \in P$ and $\sigma \in N$ the cross product $\rho \times \sigma = 0$ equals zero. Otherwise, we say that T (or π) is* nonlinear*.*

Notice that for $\gamma(p)$ big enough (bigger than $c_{max}n$ where c_{max} is the largest constant occuring in C), the sign of the effect of a simple loop does not depend on $\gamma(p)$. Hence the distinction is well defined and does not depend on γ for all but finitely many cases. By modifying the number of traversals of individual loops we will show that linear runs are bounded (Lemma 10.9) and that nonlinear runs occur only bounded number of times (Lemma 10.12).

10.7. Linear Runs

We now show that linear runs are bounded by a multiple of $\gamma(p)$. For loops ρ and σ with $\text{effect}(\rho) = a \cdot \gamma(p) + b$ and $\text{effect}(\sigma) = c \cdot \gamma(p) + d$ the constraint $\rho \times \sigma = 0$ implies that the vectors $(a, b) \in \mathbb{N}^2$ and $(c, d) \in \mathbb{N}^2$ are collinear. Hence, there is $q = (q_1, q_2) \in \mathbb{Q}^2$ such that $(a, b) = k_1 \cdot q$ and $(c, d) = k_2 \cdot q$ for some $k_1, k_2 \in \mathbb{N}$. Hence, all loops can be expressed as $k \cdot q$ (see Figure 14). By using this observation and modifying the number of traversals of loops s_i, we show that linear runs are bounded.

Lemma 10.9. *Assume $\gamma(p) > c_{max}n$. Let $\pi : (s, x) \to (t, y)$ be a linear run in C^γ such that* $\text{counter}(\pi) > 2\gamma(p)$. *Then there is an equivalent run $\pi' \sim \pi$ such that*

$$2\gamma(p) < \text{counter}(\pi') \leq \max(x, y) + V\gamma(p)$$

for some constant $V \in \mathbb{N}$ depending only on C (and not on γ).

Proof. Let π be factored as $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3$ where $\pi_2 = \alpha^a \to (f + \sum s_i p_i + \sum t_i n_i) \to \beta^b$ where p_i's are positive loops and n_i's are negative loops. Then Lemma 10.7 implies that $\text{counter}(\pi_1), \text{counter}(\pi_3) \leq \max(x, y) + n\gamma(p)$.

Let S be the set of loops in π_2 and take a loop $\sigma \in S$. Then we can uniquely write $\text{effect}(\sigma) = g\gamma(p) + h$ for some $g, h \in \mathbb{Z}$. Consider the mapping $\rho : S \to \mathbb{Z}^2$ which sends σ to the point (g, h), i.e., $\rho(\sigma) = (g, h)$. Since $\sigma \times \tau = 0$ for all loops $\sigma, \tau \in S$, the mapping ρ sends all loops to a single line. In particular, there is a rational point $q = (q_1, q_2) \in \mathbb{Q}^2$ such that for every $\sigma \in S$ there is an integer $k \in \mathbb{Z}$ such that $\rho(\sigma) = kq$. Thus, $\text{effect}(\sigma) = k(q_1\gamma(p) + q_2)$. We think of all the loops of S as being a multiple of $q_1\gamma(p) + q_2$. By a slight abuse of notation, we use $q = q_1\gamma(p) + q_2$ and write $\sigma = k \cdot q$. Note that q depends only on C (and not on γ).

We can ignore the order in which transitions are traversed in π_2. Thus, we modify the number of traversals of individual loops in π_2 while keeping the net effect of the run unchanged. Write $\beta = xq$ and $n_i = y_i q$ for some $x, y_i \in \mathbb{N}$. If $t_i \geq x$ then

$$\begin{aligned}
(t_i - x)\,\text{effect}(n_i) + (y_i + b)\,\text{effect}(\beta) &= (t_i - x)\,\text{effect}(y_i q) + (y_i + b)\,\text{effect}(xq) \\
&= t_i y_i\,\text{effect}(q) - x y_i\,\text{effect}(q) + y_i x\,\text{effect}(q) + bx\,\text{effect}(q) \\
&= t_i\,\text{effect}(y_i q) + b\,\text{effect}(xq) \\
&= \text{effect}(t_i n_i) + \text{effect}(b\beta)
\end{aligned}$$

So if $t_i > x$ then, by Lemma 10.5, there is a run π' of the form $\pi' = \pi(t_i \leftarrow t_i - x, b \leftarrow b + y_i)$ such that $\text{counter}(\pi') > 2\gamma(p)$ and $\text{effect}(\pi') = \text{effect}(\pi)$. Repeating the construction if necessary, we can assume that $t_i \leq x$ for all $n_i \in N$. Thus, we just bounded the number of traversals of negative loops by $t_i \leq x$.

Similarly, write $\alpha = yq$ and $p_i = z_i q$ for some $a, z_i \in \mathbb{N}$. Then if $s_i > y$ then, by Lemma 10.5, there is a run π' of the form $\pi(a \leftarrow a + z_i, s_i \leftarrow s_i - y)$ such that $\text{effect}(\pi'') = \text{effect}(\pi)$ and $\text{counter}(\pi'') > 2\gamma(p)$. Repeating the construction if necessary, we can assume that $s_i \leq y$ for all $p_i \in P$. Thus, we just bounded the number of traversals of positive loops by $s_i \leq y$.

Denote $f + \sum s_i p_i + \sum t_i n_i$ by τ. Then we have

$$|\,\text{div}(\tau)| \;=\; \text{div}(f + \sum s_i p_i + \sum t_i n_i) \leq n\gamma(p) + |S|y \cdot \text{div}(q) + |S|x \cdot \text{div}(q) \leq K\gamma(p)$$

for some constant K depending on S, x and y. Notice that $\text{div}(q) \leq n\gamma(p)$. Further, $|S|$ as well as x and y depend only on C (and not on γ).

Now, if $(a - x)\,\text{effect}(\alpha) \geq K\gamma(p) \geq \text{div}(\tau)$, then let $\pi''' = \pi_1 \to \alpha^{a-x} \to f + \sum_i k_i s_i \to \beta^{b-y} \to \pi_3$. By the assumption, π''' is a valid path and $\text{effect}(\pi''') = \text{effect}(\pi)$ and $\text{counter}(\pi''') > 2\gamma(p)$. Hence, by repeated application if necessary, we can assume that $\text{div}(\alpha^a) \leq K\gamma(p) + x\,\text{div}(\alpha) \leq K\gamma(p) + xn\gamma(p)$ as α is a simple loop. Therefore, $\text{div}(\alpha^a) \leq K'\gamma(p)$ for some constant $K' \in \mathbb{N}$ depending only on C. Hence, we calculate

$$\begin{aligned}
\text{counter}(\pi) &\leq \max \text{counter}(\pi_1\pi_3) + \text{div}(\pi_2) \\
&\leq \max(x, y) + n\gamma(p) + \text{div}(\alpha^a) + |\,\text{div}(\tau)| + \text{div}(\beta) && \{\text{Lemma 10.7}\} \\
&\leq \max(x, y) + n\gamma(p) + K'\gamma(p) + K\gamma(p) + n\gamma(p) \\
&\leq \max(x, y) + K''\gamma(p)
\end{aligned}$$

for some constant $K'' \in \mathbb{N}$ depending only on C (and not on γ). □

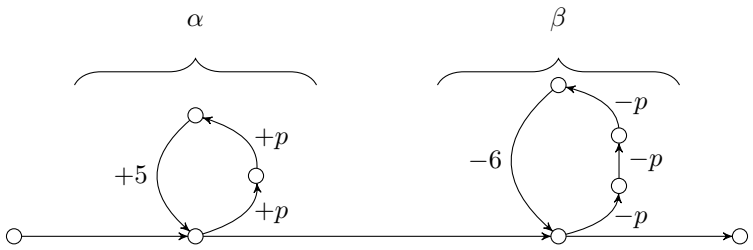


Figure 15: Counter machine with nonlinear runs. The automaton has two loops. A positive loop α with $\text{effect}(\alpha) = 2p+5$ and a negative loop β with $\text{effect}(\beta) = -3p-6$. Notice that $\alpha \times \beta = -2\cdot 5-(-3\cdot 5) = 3$ and that $\text{effect}(3\alpha)+\text{effect}(2\beta) = 3 = \alpha \times \beta$.

10.8. Nonlinear Runs

The last class of runs are the nonlinear runs. In general, we cannot bound nonlinear runs. However, we show that we can always assume that the number of nonlinear subruns in C^γ can always be bounded by a constant depending only on C (and independent of γ).

Consider a nonlinear run from configuration (s, x) to configuration (t, y). By definition, there is a positive loop ρ and a negative loop σ in the run such that $\rho \times \sigma \neq 0$. By modifying the number of times ρ and σ are traversed, we obtain runs from (s, x) to $(t, y \pm L)$ for some $L \in \mathbb{N}$ depending only on $\rho \times \sigma$ (see Figure 15). Repeating the process, we build nonlinear runs starting in (s, x) and finishing in configurations $(t, y + kL)$ for almost all valid $k \in \mathbb{Z}$.

Lemma 10.10. *Assume $\gamma(p) > c_{max}n$. Let T be a nonlinear type. Then there exists a natural number $L(T) \leq n^4$ depending only on T such that the following property holds. For states $s, t \in C$ and counter values $x, y \in \mathbb{N}$,*

- *if there is a run $\pi : (s, x) \to (t, y)$ of type T in C^γ such that $\text{counter}(\pi) > 2\gamma(p)$ and $x, y \leq 3\gamma(p)$*
- *then for every $k \in \mathbb{Z}$ with $y + kL(T) > 2\gamma(p)$ there is a run $\pi' : (s, x) \to (t, y + kL(T))$ in C^γ with $\text{counter}(\pi') > \gamma(p)$.*

Proof. Write $\pi = \pi_1 \cdot \pi_2 \cdot \pi_3 = \pi_1 \to \alpha^a \to f + \sum k_i s_i \to \beta^b \to \pi_3$. By assumption, π_2 contains a positive loop ρ and a negative loop σ such that $\rho \times \sigma \neq 0$. Write $\text{effect}(\rho) = a\gamma(p) + b$ and $\text{effect}(\sigma) = c\gamma(p) + d$. Then $c \leq 0 \leq a$ and $a \cdot \text{effect}(\sigma) - c \cdot \text{effect}(\rho) = a(c\gamma(p) + d) - c(a\gamma(p) + b) = ad - cb = \rho \times \sigma$. Hence, $a \cdot k \cdot \text{effect}(\sigma) - c \cdot k \cdot \text{effect}(\rho) = k \cdot \rho \times \sigma$ for any $k \in \mathbb{N}$. Denote $\rho \times \sigma$ by P and suppose that path π takes the loops ρ and σ exactly r and s times, respectively.

First, assume that $P > 0$. Then for any $k \in \mathbb{N}$, by Lemma 10.5, there is a path π_k of the form $\pi_k = \pi(s \leftarrow s + ak, r \leftarrow r + |c|k)$ such that $\text{counter}(\pi_k) > 2\gamma(p)$ and $\text{effect}(\pi_k) = \text{effect}(\pi) + kP$. Thus, $\text{counter}(\text{last}(\pi_k)) = y + kP$ and so by changing the number of traversals of ρ and σ we can increase the counter by an arbitrary multiple of P.

Recall that $\text{effect}(\sigma) < 0$. Therefore, there exists the largest natural number $q \in \mathbb{N}$ such that $qP + \text{effect}(\sigma) < 0$. Hence, $-P \leq qP + \text{effect}(\sigma) < 0$. If we denote $qP + \text{effect}(\sigma)$ by R then we have:

$$q\big[a \cdot \text{effect}(\sigma) - c \cdot \text{effect}(\rho)\big] + \text{effect}(\sigma) = q(\rho \times \sigma) + \text{effect}(\sigma) = qP + \text{effect}(\sigma) = R.$$

For $k \in \mathbb{N}$ suppose that $y + kR > 2\gamma(p)$. Then as $3\gamma(p) > y$ it holds that $kR > -\gamma(p)$. Therefore, by Lemma 10.5, there is a path $\tau_k = \pi(s \leftarrow s + k(qa + 1), r \leftarrow r + |c|qk)$ such that $\text{effect}(\tau_k) = \text{effect}(\pi) + kR$ and $\text{counter}(\tau_k) + kR > 2\gamma(p) + kR > \gamma(p)$. Hence, $\text{counter}(\text{last}(\tau_k)) = y + kR$.

So let $L = \text{lcm}(P, |R|)$ be the least common multiple of P and $|R|$. Then $L \leq PR \leq P^2 \leq n^4$ and by above, for every $k \in \mathbb{Z}$ such that $y + KL > 2\gamma(p)$ there is a valid run from (s, x) to $(t, y + kL)$ in C^γ.

The case $P = \rho \times \sigma < 0$ is symmetric. □

Now, there are only $L(T)$ equivalence classes modulo $L(T)$. So consider nonlinear runs $\rho_1, \rho_2, \rho_3, \ldots$ ending in configurations $(t, y_1), (t, y_2), (t, y_3)$, etc. If some runs ρ_i, ρ_j for $i < j$ finish in the same equivalence class modulo $L(T)$ (That is, we have $y_i \equiv y_j \bmod L(T)$) then, using the above lemma, there is a run from $\text{first}(\rho_i)$ to $\text{last}(\rho_j)$. And so we can skip all ρ_k for $i < k < j$.

First we define the notion of runs starting and finishing in the same configurations.

Definition 10.11. *Two runs π, π' are equivalent, written $\pi \sim \pi'$, if* $\text{first}(\pi_1) = \text{first}(\pi')$ *and* $\text{last}(\pi) = \text{last}(\pi')$.

Then we can state the lemma on the rerouting of runs as follows.

Lemma 10.12. *Assume $\gamma(p) > c_{max}n$. Given a run π in C^γ written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_k \to \tau_k$ where π_i's are nonlinear runs and* $\text{counter}(\pi_i) > 2\gamma(p)$ *and* $\text{counter}(\text{first}(\pi_i)), \text{counter}(\text{last}(\pi_i)) \leq 3\gamma(p)$. *Then there is an equivalent run $\pi' \sim \pi$ such that π' factors as:*

$$\tau_0 \to \pi'_1 \to \tau_{f(1)} \to \pi'_2 \to \tau_{f(2)} \to \ldots \to \pi'_l \to \tau_{f(l)}$$

where

- $\pi'_i : \text{first}(\pi_{f(i-1)}) \to \text{last}(\pi_{f(i)}))$,
- $\text{counter}(\pi'_i) \geq \gamma(p)$,
- $f : [1 \ldots l] \to [1 \ldots k]$ *is a strictly increasing function,*
- $l \leq Dn^5$ *where D is the number of different types.*

Proof. Write the last configuration $\text{last}(\pi_i)$ of π as $\text{last}(\pi_i) = (t_i, y_i)$. Let L be the value from Lemma 10.10 associated with the type of π_1. According to Lemma 10.10, if there is i such that $t_i = t_1$ and $y_1 \equiv y_i \bmod L$ then there is a run π'_1 from $\text{first}(\pi_1)$ to (t_i, y_i). Set $f(1)$ to be the largest such i.

Then consider $\pi_{f(1)+1}$ and let L' be the value from Lemma 10.10 associated with its type. Now, if there is i' such that $t_{i'} = t_{f(1)+1}$ and $y_{f(1)+1} \equiv y_{i'} \bmod L'$ then, by Lemma 10.10, there is a run π'_2 from $\text{first}(\pi_{f(1)+1})$ to $(t_{i'}, y_{i'})$. Set $f(2)$ to be the largest such i'.

Repeat this process, until eventually we set $f(l) = k$ for some l. Now, for each type T and each state t the process is repeated at most $L(T)$ times (once for each equivalence class). Hence, $l \leq Dn \max_T L(T) \leq Dn^5$. □

10.9. Factoring of Runs

We now combine the above results and show that any run in C^γ has always an equivalent factoring of the desired form (1) described on page 29.

Theorem 10.13. *Assume $\gamma(p) > c_{max}n$. Let π be a run in C^γ. Then there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that π can be written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ where $t \leq L$ and* $\text{counter}(\tau_i) \leq K\gamma(p)$ *and* $\gamma(p) < \text{counter}(\pi_i)$ *and* $2\gamma(p) < \text{counter}(\text{first}(\pi_i)), \text{counter}(\text{last}(\pi_i)) \leq 3\gamma(p)$ *for every i.*

Proof. Let π be an accepting run in C^γ and write $\pi = \sigma_0 \to \pi_1 \to \sigma_1 \to \pi_2 \to \sigma_3 \to \ldots \to \pi_k \to \sigma_k$ such that $\text{counter}(\sigma_i) \leq 2\gamma(p) < \text{counter}(\pi_i)$ for every i. Since every transition of C increases the counter by at most $\gamma(p)$, we have $2\gamma(p) < \text{counter}(\text{first}(\pi_i)) \leq 3\gamma(p)$.

Let Π be the collection of all nonlinear subruns π_i. Applying Lemma 10.12 to π gives us a split: $\pi = \tau'_0 \to \pi'_1 \to \tau'_1 \to \ldots \to \pi'_t \to \tau'_t$ such that $\text{counter}(\pi'_i) \geq \gamma(p)$ and each τ'_i is a concatenation of some consecutive τ_j's and π_k's where $\pi_k \notin \Pi$. Further, $t \leq 3Dn^5$.

Thus, τ'_i can be written as $\tau'_i = \rho_1 \ldots \rho_r$ such that each ρ_i equals either (i) t_j for some j or (ii) π_j for some linear π_j or (iii) π_j for some nonpumpable π_j.

In each case, we can bound $\text{counter}(\rho_i)$ as follows.

- If $\rho_i = \tau_j$ for some j then $\text{counter}(\tau'_i) = \text{counter}(\tau_j) \leq 2\gamma(p)$.

- If $\rho_i = \pi_j$ for some $\pi_j \notin \Pi$ and π_j is linear then, by Lemma 10.7, $\text{counter}(\tau_i') \leq \text{counter}(\pi_j) = (3+n)\gamma(p)$.
- If $\rho_i = \pi_j$ for some $\pi_j \notin \Pi$ and π_j is not pumpable then, by Lemma 10.9, $\text{counter}(\tau_i') \leq \text{counter}(\pi_j) \leq \max(x,y) + V\gamma(p)$.

So take K to be the maximum of the constants appearing above. □

10.10. Decision Procedures

We now show how to use the factoring of runs from Theorem 10.13 to decide the emptiness problem for C. Recall that the above results assume that $\gamma(p) > c_{max}n$ to ensure that the notion of positive and negative loops is well defined. Thus, the first step in the decision procedure is to instantiate p at $p = 0, \ldots, c_{max}n$ and check the existence of an accepting run (e.g., using Lemma 5.1) in the resulting nonparametric machines.

Suppose that for no value of $p = 0, \ldots, c_{max}n$ we found an accepting run. Then we can assume that $\gamma(p) > c_{max}n$. We now give ∃PAD formulae encoding the existence of an assignment γ and a factoring from Theorem 10.13 of an accepting run.

Theorem 10.14. *Given C and states $s, t \in C$, the $J(C,s,t) = \{(x,y,q) \mid (s,x) \to^* (t,y)$ in C^γ where $\gamma(p) = q$ and $q > c_{max}n\}$ is* ∃PAD *definable.*

Proof. We encode the existence of accepting runs from Theorem 10.13. For runs τ_i, we use Theorem 9.1 to express reachability up to a given multiple of $\gamma(p)$. For runs π_i we use Lemma 10.1 and express them as runs in the parametric one-counter machine $C_{>p}$—reachability in which is ∃PAD definable (Lemma 5.1).

So, given l—the length of the factoring in Theorem 10.13 and states $u_0, u_1, \ldots, u_l, v_0, v_1, \ldots, v_l \in C$—the initial and final states of τ, respectively, consider the following formula:

$$J'(C,s,t,l,\overrightarrow{u},\overrightarrow{v})(x,y,p) = \exists \overrightarrow{x}, \overrightarrow{y}. \quad \bigwedge_{i=1\ldots l} G(C,s_i,t_i,K)(x_i,y_i,p) \;\bigwedge_{i=1\ldots l} \text{Reach}(C_{>p},t_{i-1},s_i)(y_{i-1},x_i,p) \;\bigwedge_{i=1\ldots l} (2p < x_i < 3p \;\wedge\; 2p < y_i < 3p) \;\bigwedge G(C,s_0,t_0)(x,y_0,p) \;\wedge\; y_l = y \;\bigwedge p > c_{max}n$$

The formula asserts the existence of a particular factoring from Theorem 10.13. Since there are only finitely many possible values for l, $\overrightarrow{u}$ and $\overrightarrow{v}$, the result follows as ∃PAD sets are closed under finite union. □

Since the satisfiability of ∃PAD-definable sets is decidable, we have:

Theorem 10.15. *The emptiness problem is decidable for simple parametric bounded one-counter machines with a single parameter.*

10.10.1. Elimination of '$\equiv 0 \bmod c$' and '$+[0,p]$' Transitions

In the previous section, we showed decidability of the emptiness problem for simple parametric bounded one-counter machines. However, recall that our goal is to show decidability for parametric bounded one-counter machines as the emptiness problem for this class is equivalent (Theorem 6.10 and Theorem 6.7) to the emptiness problem for parametric timed automata with two parametric clocks. We now show how to extend our techniques to support '$\equiv 0 \bmod c$' and '$+[0,p]$' transitions in the case of a single parameter.

10.10.2. Elimination of '$\equiv 0 \bmod c$' Transitions

First suppose that C contains '$\equiv 0 \bmod c$' transitions but no '$+[0,p]$' transitions. Applying Theorem 10.14 to C_D's from Theorem 9.5 shows that the reachability for machines with '$\equiv 0 \bmod c$' transitions is ∃PAD definable. Assuming $\gamma(p) > c_{max}n$ we have:

Theorem 10.16. *Given C with '$\equiv 0 \bmod c$' transitions and states $s, t \in C$. The set $K(C,s,t) = \{(x,y,q) \in \mathbb{N}^3 \mid (s,x) \to^* (t,y)$ in C^γ where $\gamma(p) = q$ and $q > c_{max}n\}$ is* ∃PAD *definable.*

Proof. According to Lemma 9.5 from page 26, it suffices to consider machines C_D for all conceivable D's. For a fixed $c \in \mathbb{N}$ the $\exists$PAD formula $\varphi_c(x,y) = (\exists q.c \cdot q + y = x) \wedge y < c$ asserts that $x \equiv y \mod c$.

So let $\{c_1, \ldots, c_r\}$ be the set of constants appearing in '$\equiv 0 \bmod c$' transitions in C. Given $D \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$, we construct the formula $K'(C,s,t,D)(x,y,p)$ asserting that D is consistent with p and that $(s,x) \to^* (t,y)$ in C_D:

$$K'(C,s,t,D)(x,y,p) = \exists \overrightarrow{v}(r), \overrightarrow{w}(r) \quad . \quad H(C_D, s \times v, t \times w)(x,y,p) \\ \bigwedge_i [(x \equiv v(i) \mod c_i) \wedge (y \equiv w(i) \mod c_i) \wedge (p \equiv D(i) \mod c_i)] \\ \wedge \; p > c_{max} n$$

Since $\exists$PAD definable sets are closed under finite union and there are only finitely many possible D's, we get the desired result. □

10.10.3. Elimination of '$+[0,p]$' Transitions

Now suppose that C is a parametric bounded one-counter machine C containing '$\equiv 0 \bmod c_i$' as well as '$+[0,p]$' transitions. Recall that in Lemma 9.6 we showed that if there is an accepting run in C^γ then there is an accepting run where in all but a finitely many cases the counter updates by '$+[0,p]$' transitions lie in the set $\{0, \ldots, R, \gamma(p) - R, \ldots, \gamma(p)\}$ where $R = \mathrm{lcm}(c_1, \ldots, c_r)$ is the least common multiple of all constants appearing in '$\equiv 0 \bmod c_i$' transitions.

The result, analogously to Theorem 9.7, can be directly used to show that the existence of an accepting run C^γ can be specified by a $\exists$PAD formula. Assuming $\gamma(p) > c_{max} n$ we have:

Theorem 10.17. *Given C with '$\equiv 0 \bmod c$' and '$+[0,p]$' transitions and states $s,t \in C$ then the set $L(C,s,t) = \{(x,y,q) \in \mathbb{N}^3 \mid (s,x) \to^* (t,y)$ in C^γ where $\gamma(p) = q$ and $q > c_{max} n\}$ is $\exists$PAD definable.*

Proof. Let Z be the machine obtained from C by replacing each '$+[0,p]$' transition by $2R+2$ transitions: $+0, +1, \ldots, +R, +p-R, \ldots, +p$. Then, by Lemma 9.6, we can assume that an accepting run π in C^γ can be factored as $\pi = \pi_0 \to \pi_1 \to \ldots \to \pi_v$ where each π is a run in Z^γ and there is a '$+[0,p]$' transition between π_i and π_{i+1}.

Given $v \leq nR$ and states $s = s_1, s_2, \ldots, s_v, t_1, \ldots, t_{v-1}, t_v = t$ the formula

$$L'(C,s,t,v,\overrightarrow{s},\overrightarrow{t})(x,y,p) = \exists \overrightarrow{x}, \overrightarrow{y}. \quad x_1 = x \wedge y_v = y \\ \bigwedge_{i=1\ldots v} K(Z, s_i, t_i)(x_i, y_i, p) \\ \bigwedge_{i=1\ldots v-1} 0 \leq y_{i+1} - x_i \leq p \\ \bigwedge_{i=1\ldots v-1} E(t_i, \text{`}+[0,p]', s_{i+1}) \\ \wedge \quad p > c_{max} n$$

asserts that such a factoring of length v exists where $\mathrm{first}(\pi_i) = (s_i, x_i)$ and $\mathrm{last}(\pi_i) = (t_i, y_i)$. The predicate $E(q, \text{`}+[0,p]', q')$ encodes that there is a '$+[0,p]$' transition between q and q' in C.

Notice there are only finitely many different values for $v' \leq nR$ and $\overrightarrow{s}$ and $\overrightarrow{t}$. Since $\exists$PAD sets are closed under finite union, the result follows. □

Since $\exists$PAD satisfiability is decidable, we have:

Theorem 10.18. *The emptiness problem for parametric bounded one-counter machines with a single parameter is decidable.*

10.10.4. Simple Programs

We now show how to use the developed theory to show decidability for simple programs—a model introduced by O. Ibarra *et al.* in 1990's [14]—the emptiness problem of which is still an open problem. A simple program is a simple parametric bounded one-counter machine (no '$+[0,p]$' or '$\equiv 0 \bmod c$' edges)

N with the addition that the counter can become negative. For example, subtracting 5 when the counter equals 2 is valid and yields the counter equal to -3.

In case N has a single parameter p, we show that the emptiness problem for N is decidable. Suppose that there is an assignment γ and an accepting run in N^γ. Then consider a shortest accepting run π in N^γ. Split π into positive and negative subruns: $\pi = \pi_1 \to \nu_1 \to \ldots \pi_k \to \nu_k$ for some k where $\text{counter}(\nu_i) \leq 0 \leq \text{counter}(\pi_i)$ for each i.

Note that Theorem 10.13 from the previous section applies to positive subruns. Hence, there are $K_1, L_1 \in \mathbb{N}$ such that all but L_1 subruns π_i satisfy, $\text{counter}(\pi_i) \leq K_1\gamma(p)$.

By multiplying the counter by -1, we can think of $-1 \cdot \nu_i$'s as runs in a simple parametric bounded one-counter machine. Similarly, there are $K_2, L_2 \in \mathbb{N}$ such that all but L_2 subruns ν_i satisfy, $\text{counter}(\nu_i) \geq -K_2\gamma(p)$.

Taking $K = \max(K_1, K_2)$ and $L = L_1 + L_2$ we get the following:

Theorem 10.19. *Let π be a run in N^γ. Then there are $K, L \in \mathbb{N}$ depending only on C (and not on γ) such that π can be written as $\pi = \tau_0 \to \pi_1 \to \tau_1 \to \pi_2 \to \tau_2 \to \ldots \to \pi_t \to \tau_t$ where $t \leq L$ and $|\text{counter}(\tau_i)| \leq K\gamma(p)$ and either $\gamma(p) < \text{counter}(\pi_i)$ or $-\gamma(p) > \text{counter}(\pi_i)$ for every i. Further $2\gamma(p) < |\text{counter}(\text{first}(\pi_i))|, |\text{counter}(\text{last}(\pi_i))| \leq 3\gamma(p)$ for every i.*

Since the inverse $-1 \cdot \nu_i$ is a run in a parametric one-counter machine, similarly to Theorem 10.14, a factoring of the above form can be verified by a $\exists$PAD formula. Thus, we obtain:

Theorem 10.20. *Given a simple program N and states $s, t \in C$, the set $H_{sp}(C, s, t) = \{(x, y, q) \mid (s, x) \to^* (t, y)$ in N^γ where $\gamma(p) = q\}$ is $\exists$PAD definable.*

Proof. Given N and states $s, t \in C$, the set $H_{sp}(C, s, t) = \{(x, y, q) \mid (s, x) \to^* (t, y)$ in N^γ where $\gamma(p) = q\}$ is $\exists$PAD definable.

Consider the machine $N_{<-p}$, which is obtained from N by removing all $\geq c, \geq p$ edges and replacing all $\leq c, \leq p$ edges by $+0$ edges. Finally, let M be obtained from N by multiplying all updates by -1. For example, edge $+p$ in $N_{<-p}$ becomes $-p$ in M.

Given l – the length of the factoring in Theorem 10.19 and states $u_0, u_1, \ldots, u_l$, $v_0, v_1, \ldots, v_l \in C$—the initial and final states of τ, respectively, consider the following formula:

$$
\begin{aligned}
H'_{sp}(C, s, t, l, \overrightarrow{u}, \overrightarrow{v})(x, y, p) = \exists \overrightarrow{x}, \overrightarrow{y}.\quad & \bigwedge_{m=1\ldots l} G(C, s_m, t_m, K)(x_m, y_m, p) \\
& \bigwedge_i \text{Reach}(N_{>p}, t_{i-1}, s_i)(y_{i-1}, x_i, p) \\
& \bigwedge_j \text{Reach}(M, t_{j-1}, s_j)(-y_{j-1}, -x_j, p) \\
& \wedge\ G(C, s_0, t_0)(x, y_0, p) \ \wedge\ y_l = y \\
& \bigwedge_i (2p < x_i < 3p \ \wedge\ 2p < y_i < 3p) \\
& \bigwedge_j (2p < -x_j < 3p \ \wedge\ 2p < -y_j < 3p)
\end{aligned}
$$

where index i ranges over π_i with $\text{counter}(\pi_i) > \gamma(p)$ and index j ranges over π_j with $\text{counter}(\pi_k) < -\gamma(p)$. The formula asserts the existence of a particular factoring from Theorem 10.19. Since there are only finitely many possible values for $l, \overrightarrow{u}, \overrightarrow{v}$, the result follows as $\exists$PAD sets are closed under finite union. □

11. Conclusion

In this manuscript we studied the decidability of the emptiness problem for parametric timed automata. See Table 2 for the currently best-known complexities. Even though we were unable to prove the case of two parametric clocks in full generality, we believe that the reduction given to one-counter machines might play an important step in resolving the problem as was the case in nonparametric settings where a similar reduction [11] led to the resolution of precise complexity for the reachability problem in (nonparametric) two-clock timed automata [7]. See Figure 12 summarizing the relationship between the classes.

	One Parametric Clock	Two Parametric Clocks		Three Parametric Clocks
		One Parameter	General	
Upper Bound	2NEXP	Decidable	Open	Undecidable
Lower Bound	NEXP-hard	$\mathsf{PSPACE}^{\mathsf{NEXP}}$		

Table 2: Complexity of the emptiness problem for various classes of parametric timed automata

For one parametric clock we showed the emptiness problem decidable in 2NEXP time (Theorem 6.5). We have also showed a lower bound of NEXP-hard (Theorem Theorem 6.6). However, the upper bound is obtained by reduction to ∃PAD satisfiability, the precise complexity of which is still unknown, only known to lie between NP-hard and NEXP. Thus, it is possible that the lower bound is optimal and further progress depends on the developments in the ∃PAD-satisfiability problem.

For parametric timed automata with two parametric clocks, however, the situation is more complicated. We established a $\mathsf{PSPACE}^{\mathsf{NEXP}}$ lower bound. It is conceivable that the lower bound can be further improved to EXPSPACE. In proof of the lower bound (Theorem 6.11), we employed polynomially many nonparametric clocks to store polynomially many bits. Hence polynomially many clocks can encode numbers exponentially large in magnitude, which could then be used to point into exponential memory. However, we were unable to devise a way of storing exponential many bits in two parametric clocks which would then yield an EXPSPACE lower bound. The resolution of the decidability in two parametric-clocks case is the still open and is unknown already for two parameters.

It is not clear whether the techniques developed in the closing sections of the manuscript generalise to such multiparametric settings.

The decision procedures for parametric one-counter machines are of high complexity, on the other hand, the best lower bound for the emptiness problem for parametric bounded one-counter machines we are aware of (PSPACE) occurs already in nonparametric setting [7]. Can the lower bound be improved by employing parameters?

Finally, note that the results presented in this chapter give ∃PAD characterisation of the reachability relation in the respective classes of parametric bounded one-counter machines. Therefore, it is conceivable that the results can be used to solve other problems than plain reachability, e.g., model checking or the existence of a Büchi path in parametric bounded-one counter machines or the corresponding parametric timed automata.

12. Bibliography

[1] Eric Allender, Michal Koucký, Detlef Ronneburger, and Sambuddha Roy. The pervasive reach of resource-bounded kolmogorov complexity in computational complexity theory. *J. Comput. Syst. Sci.*, 77(1):14–40, January 2011.

[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*, 1993.

[3] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, April 1994.

[4] A.P. Bel'tyukov. Decidability of the universal theory of natural numbers with addition and divisibility. *Journal of Soviet Mathematics*, 14(5):1436–1444, 1980.

[5] Daniel Bundala and Joël Ouaknine. Advances in parametric real-time reasoning. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 123–134. Springer, 2014.

[6] L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208 – 213, 2007.

[7] J. Fearnley and M. Jurdziński. Reachability in two-clock timed automata is PSPACE-Complete. In *Proceedings of the 40th International Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'13, 2013.

[8] S. Göller, C. Haase, J. Ouaknine, and J. Worrell. Model checking succinct and parametric one-counter automata. In *ICALP'13*, volume 6199 of *Lecture Notes in Computer Science*. Springer, 2010.

[9] C. Haase. *On the Complexity of Model Checking Counter Automata*. PhD thesis, University of Oxford, 2012.

[10] C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR'09*, volume 5710 of *Lecture Notes in Computer Science*. Springer, 2009.

[11] C. Haase, J. Ouaknine, and J. Worrell. On the relationship between reachability problems in timed and counter automata. In *RP*, volume 7550 of *Lecture Notes in Computer Science*. Springer, 2012.

[12] T. A. Henzinger, Z. Manna, and A. Pnueli. What good are digital clocks? In *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*. Springer, 1992.
[13] T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*. 2001.
[14] O. H. Ibarra, T. Jiang, N. Q. Trân, and H. Wang. New decidability results concerning two-way counter machines and applications. In *ICALP'93*, volume 700 of *Lecture Notes in Computer Science*. Springer, 1993.
[15] A. Jovanovic, D. Lime, and O. H. Roux. Integer parameter synthesis for timed automata. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, 2013.
[16] P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for AC-like equational theories with homomorphisms. In *Research Report LSV-04-16*, LSV, ENS de Cachan, 2004.
[17] F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model checking timed automata with one or two clocks. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 387–401. Springer Berlin Heidelberg, 2004.
[18] Antonial Lechner, Joël Ouaknine, and James Worrell. On the complexity of linear arithmetic with divisibility (submitted).
[19] L. Lipshitz. The Diophantine Problem for Addition and Divisibility. *Transactions of The American Mathematical Society*, volume 235, 1978.
[20] L. Lipshitz. Some remarks on the diophantine problem for addition and divisibility. *Proceedings of the Model Theory Meeting*, volume 33, 1981.
[21] J. S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *Hybrid Systems: Computation and Control*, volume 1790 of *Lecture Notes in Computer Science*. 2000.
[22] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.
[23] Joël Ouaknine and James Worrell. Universality and language inclusion for open and closed timed automata. In *Proceedings of the 6th International Conference on Hybrid Systems: Computation and Control*, HSCC'03, pages 375–388, Berlin, Heidelberg, 2003. Springer-Verlag.