Advances in Parametric Real-Time Reasoning

Daniel Bundala and Joël Ouaknine

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Abstract. We study the decidability and complexity of the reachability problem in parametric timed automata. The problem was introduced 20 years ago by Alur, Henzinger, and Vardi in [1], where they showed decidability in the case of a single parametric clock, and undecidability for timed automata with three or more parametric clocks.
By translating such problems as reachability questions in certain extensions of parametric one-counter machines, we show that, in the case of two parametric clocks (and arbitrarily many nonparametric clocks), reachability is decidable for parametric timed automata with a single parameter, and is moreover $\text{PSPACE}^{\text{NEXP}}$-hard. In addition, in the case of a single parametric clock (with arbitrarily many nonparametric clocks and arbitrarily many parameters), we show that the reachability problem is NEXP-complete, improving the nonelementary decision procedure of Alur *et al.*

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 [1]: 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?

In this framework, 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). The decision procedure given in [1] however 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 [1], with hardly any progress (partial or otherwise) that we are aware of in the intervening period. Alur *et al.* showed that this problem subsumes the question of reachability in Ibarra's "simple programs" [11], also open for over 20 years, as well as the decision problem for a fragment of Presburger arithmetic with divisibility.

Our main results are as follows: (i) We show that, in the case of two parametric clocks (and arbitrarily many nonparametric clocks), reachability is decidable for parametric timed automata with a *single* parameter. Furthermore, we establish a $\text{PSPACE}^{\text{NEXP}}$ lower bound on the complexity of this problem. (ii) In the case of a single parametric clock (with arbitrarily many nonparametric clocks and arbitrarily many parameters), we show that the reachability problem is NEXP-complete, improving the nonelementary decision procedure of Alur *et al.*

Our results rest in part on new developments in the theory of one-counter machines [7], their encodings in Presburger arithmetic [6], and their application to reachability in (ordinary) timed automata [8, 4]. We achieve this by restricting our attention to parametric timed automata with *closed* (i.e., *non-strict*) clock constraints. As parameters are restricted to ranging over integers,[1] standard digitisation techniques apply [9], reducing the reachability problem over dense time to discrete (integer) time. (Alternately, our results also apply directly to timed automata interpreted over discrete time, regardless of the type of constraints used.) The restriction to integer time enables us, among others, to keep track of the values of two parametric clocks using a single counter, in effect reducing the reachability problem for timed automata with two parametric clocks to a halting problem for parametric one-counter machines.

Related Work The decidability of reachability for parametric timed automata can be achieved in certain restricted settings, for instance by bounding the allowed range of the parameters [12] or by requiring that parameters only ever appear either as upper or lower bounds, but never as both [10]: 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. The primary concern in such restricted settings is usually the development of practical verification tools, and indeed the resulting algorithms tend to have comparatively good complexity.

Miller [16] 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 [3] 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 [8], and exploited to show that reachability for (ordinary) two-clock timed automata is polynomial-time equivalent to the halting 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 paper. The reduction of [8] was used in [4] to show that halting for bounded one-counter machines, and hence reachability for two-clock timed

[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., [16, 3], discussed further below.

automata, is PSPACE-complete, solving what had been a longstanding open problem.

Finally, parametric one-counter machines in which no upper bounds are imposed on the value of the counter were studied in [7], where reachability was shown to be decidable. The techniques used in that paper make crucial use of the unboundedness of the counter and therefore do not appear applicable in the present setting.

2 Preliminaries

We now give definitions used throughout the rest of the paper. A ***timed automaton*** is a finite automaton extended with clocks; each clock measuring the time since it was last reset. A parametric timed automaton is obtained by replacing the known constants in the guards by parameters.

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. A ***parametric timed automaton*** $A = (S, s_0, C, P, F, E)$ is a tuple where S is the set of states, $s_0 \in S$ is the initial state, C is the set of clocks, P is the set of parameters, $F \subseteq S$ is the set of final states and $E \subseteq S \times S \times 2^C \times G(C, P)$ is the set of edges. An edge (s, s', R, G) is from state s to state s'. And $G(C, P)$ is the set of guards of the form $x \leq v, x \geq w$ where x is a clock and $v, w \in \mathbb{N} \cup P$. Set R specifies which clocks are reset. A clock is ***parametrically constrained*** if it is compared to a parameter in some transition. Let γ be an assignment to parameters then A^γ denotes the automaton obtained by setting each parameter $p \in P$ to $\gamma(p)$.

A ***configuration*** (s, ν) of A^γ consists of state s and function $\nu : C \to \mathbb{N}$ assigning a value to each clock. A transition exists from configuration (s, ν) to (s', ν') in A^γ ,written $(s, \nu) \to (s', \nu')$, if either there exists $t \in \mathbb{N}$ such that $\nu(c) + t = \nu'(c)$ for every clock $c \in C$ or there is an edge $e = (s, s', R, G) \in E$ such that G is satisfied for current clock values and if $c \in R$ then $\nu'(c) = 0$ and if $c \notin R$ then $\nu'(c) = \nu(c)$.

The initial clock valuation ν_0 assigns 0 to every clock. A ***run*** of a machine 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. The ***existential halting problem***, also known as parametric reachability or the emptiness problem, asks whether there exists parameter valuation γ such that A^γ has an accepting run. From here onwards, we omit "existential" and write simply "halting problem". We say that two automata A_1 and A_2 have ***equivalent halting problem*** if A_1 halts if and only if A_2 halts.

Given a run π, we use $start(\pi) = \boldsymbol{c_1}$ and $end(\pi) = \boldsymbol{c_k}$ to denote the first and the last configuration of the run, respectively. If τ is a run, we write $\pi \to \tau$ if the runs can be connected by a transition, i.e., $end(\pi) \to start(\tau)$.

A ***parametric timed 0/1 automaton*** [1] $A = (S, s_0, C, P, F, E)$ is a timed automaton such that each edge $e \in E$ is labeled by a time increment $t \in \{0, 1\}$. A transition from (s, ν) to (s', ν') is valid only if $\nu'(c) - \nu(c) = t$ for each $c \in C$ not reset by the edge.

A ***one-counter machine*** is a finite-state machine equipped with a single counter. Each edge is labelled by an integer, which is added to the counter whenever that edge is taken. The counter is required to be nonnegative at all times. E.g, subtracting $c \in \mathbb{N}$ in one transition and adding c in the next transition leaves the counter unchanged but can be performed only if the counter is at least c.

A ***bounded one-counter machine*** also allows $\leq x$ edges. Such an edge can be taken only when the counter is at most x. Reachability in these two classes of counter machines are respectively known to be NP-complete [7] and PSPACE-complete [4] if the numbers are encoded in binary.

Parametric machines are obtained by replacing the known constants by parameters. Formally, a ***parametric bounded one-counter machine*** $C = (S, s_0, F, P, E, \lambda)$ is a tuple where S is the set of states, s_0 is the initial state, $F \subseteq S$ are the final states, P is the set of parameters, $E \subseteq S \times S$ is the set of edges and $\lambda : E \to Op$ assigns an operation to each edge and has codomain Op: $\{+c, -c, +p, -p, \leq c, = c, \geq c, \leq p, = p, \geq p, +[0,p], \equiv 0 \bmod c \ : \ c \in \mathbb{N}, p \in P\}$. A ***parametric one-counter machine*** allows only operations: $\pm c, \pm p, \geq c, \geq p, = 0$. Note that parametric one-counter machines are a subclass of parametric bounded one-counter machines.

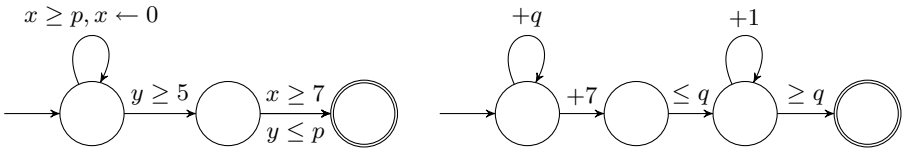

Fig. 1. A parametric timed automaton (left) and a parametric bounded one-counter machine (right). The final states are reachable if, for example, $p = 10$ and $q = 11$.

A ***configuration*** (s, x) of C consists of a state $s \in S$ and counter value $x \geq 0, x \in \mathbb{N}$. Machine C starts in state s_0 and counter equal to 0 and then takes individual edges updating the counter. We use $\mathit{counter}(s, x) = x$ to denote the counter value in a configuration. We extend the definition to runs componentwise and write $\mathit{counter}(\pi) \leq C$ (resp. $\mathit{counter}(\pi) \geq C$) if the comparison holds for every element: $\forall i \ . \ \mathit{counter}(\pi(i)) \leq C$ (resp. $\forall i \ . \ \mathit{counter}(\pi(i)) \geq C$).

Let Z be a (nonparametric) one-counter machine. For 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., $x < \mathit{counter}(\pi) < y$.

Let γ be a parameter assignment. A configuration (s', x') is reachable in one step from (s, x) (written $(s, x) \to (s', x')$) in C^γ if there exists an edge $e = (s, s') \in E$ such that

- if $\lambda(e) = \pm c, c \in \mathbb{N}$ then $x \pm c = x'$
- if $\lambda(e) = \pm p, p \in P$ then $x \pm \gamma(p) = x'$
- if $\lambda(e) = \sim c, c \in \mathbb{N}$ then $x = x'$ and $x \sim c$ where $\sim \in \{\leq, \geq\}$
- if $\lambda(e) = \sim p, p \in P$ then $x = x'$ and $x \sim \gamma(p)$ where $\sim \in \{\leq, \geq\}$
- if $\lambda(e) = +[0, p], p \in P$ then $x \leq x' \leq x + \gamma(p)$
- if $\lambda(e) = \equiv 0 \bmod c, x \in \mathbb{Z}$ then $x = x'$ and $x \equiv 0 \bmod c$

The ***existential halting problem*** asks whether there is a parameter valuation γ such that C^γ has an accepting run.

2.1 Presburger Arithmetic

Presburger Arithmetic with Divisibility is the first-order logical theory of $\langle \mathbb{N}, <, +, |, 0, 1 \rangle$. 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 . The satisfiability of $\exists$PAD formulae was shown decidable in [14, 2] and in NP [15]. 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 $x_1, \ldots x_k$ 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 [6, 7] that the reachability relation of parametric one-counter machines is $\exists$PAD definable.

Lemma 1 ([6], Lemma 4.2.2). *Given a parametric one-counter machine B and states s, t, the relation $Reach(B, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \to^* (t, y)$ in B^γ where $\gamma(p_i) = n_i\}$ is $\exists$PAD definable.*

2.2 Nonparametric Clock Elimination

Let A be a parametric timed automaton. By modifying the region construction, we show how to build a parametric timed automaton with equivalent halting problem without nonparametric clocks. 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 the value of c is always a natural number, we eliminate all nonparametric clocks by storing in the state space 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 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. Formally:

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

3 One Parametric Clock

For the rest of the section, fix a parametric timed automaton A with one parametric clock. We show how to decide the halting problem for A. By Lemma 2, there is an exponentially larger parametric 0/1 automaton B with one (parametrically constrained) clock such that A and B have equivalent halting problem.

In Lemma 4 we further show how to eliminate clock resets from B by introducing -1 transitions, thereby turning B into a parametric bounded one-counter machine. Hence, to decide the halting problem for A it suffices to decide the halting problem for a parametric bounded one-counter machine with only $-1, 0, +1$ counter updates. We establish such a result in Theorem 5, which then yields:

[2] A single formula would be logically sufficient, but would result in exponential blowup

Theorem 3. *The halting problem for parametric timed automata with one parametric clock is decidable in NEXP time.*

Decidability of the halting problem for the above class originally appeared in [1], albeit with nonelementary complexity. We give a completely different proof using one-counter machines yielding a NEXP algorithm. Later we show that the problem is also NEXP-hard. In the appendix we prove a necessary lemma:

Lemma 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 halting problem. Further, all updates in C are either $-1, 0$ or $+1$ and $|C| = O(|B|)$.*

3.1 Decidability For Counter Machines With Constant Updates

We reduced the halting problem for A to the halting 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. 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 their precise values. 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 and consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$. There is a constant $M \in \mathbb{N}$, determined in Lemma 7, such that the run π can be factored into subruns between successive parameters and subruns around individual parameters (see Fig. 2). Formally, $\pi = \pi_0 \to \pi_1 \to \pi_2 \to \cdots \to \pi_l$ such that (π_0 can be possible empty)

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

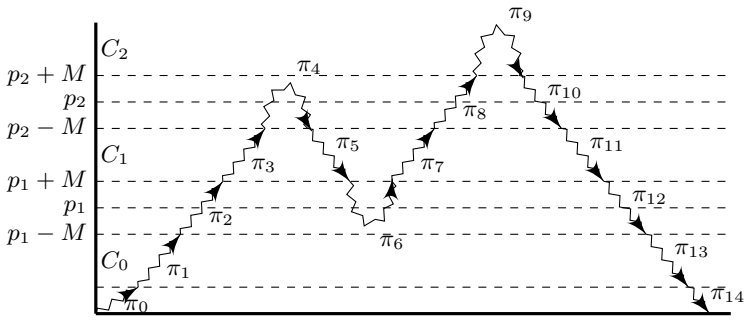

Fig. 2. Factoring of a run, which starts and finishes with the counter equal to 0. The y axis shows the counter value, hypothetical values of the parameters and their neighbourhoods. Label C_i marks the interval corresponding to counter machine C_i.

Notice that every transition in C changes the counter by at most 1. Hence, $\mathit{counter}(\mathit{start}(\pi_{2i+1})) = p_r + M + 1$ or $\mathit{counter}(\mathit{start}(\pi_{2i+1})) = p_{r+1} - M - 1$.

Thus, $start(\pi_i)$ is always of the form $start(\pi_i) = (s_i, p_{f(i)} + x)$ for some state s_i, some $|x_i| \in \{M, M+1\}$ and parameter $p_{f(i)}$. Hence, $start(\pi_i)$ is uniquely determined by the triple $(s_i, f(i), x_i)$. Similarly, $end(\pi_i)$ 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 only 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 $O(nk)$.

To show that there is a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ we guess a factoring of the above form. We shall show (justifying the choice of M) in Lemma 8 that the odd-indexed runs π_{2i+1} correspond to runs in some one-counter machine $C_{h(2i+1)}$. By Lemma 1, the existence of a run in $C_{h(2i+1)}$ is $\exists$PAD expressible as: $\varphi_{2i+1} = Reach(C_{h(2i+1)}, s_{2i+1}, t_{2i+1})(n_{f(2i+1)} + x_{2i+1}, n_{g(2i+1)} + y_{2i+1}, n_1, \ldots, n_k)$.

In Lemma 9, we show that the even-indexed runs are independent of γ, can be precomputed and the reachability relation can be hardwired into the formula. Thus, we express the existence of a particular factoring from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ as $\varphi = \bigwedge_i \varphi_{2i+1} \wedge \psi(f, g, h, \overrightarrow{s}, \overrightarrow{t}, \overrightarrow{x}, \overrightarrow{y}) \wedge \bigwedge_i (n_i + M < n_{i+1})$ where the middle term encodes that the odd- and even-indexed runs are adjacent (directly computable) and that the even-indexed runs are valid (Lemma 9). The last conjunct encodes the technical restriction $\gamma(p_i) + M < \gamma(p_{i+1})$ imposed in Lemmas 8 and 9.

The restriction is relaxed as follows. First, if the parameters are not in the increasing order $\gamma(p_i) < \gamma(p_{i+1})$ then we relabel the parameters and build the appropriate formula. If $\gamma(p_i) \leq \gamma(p_{i+1}) < \gamma(p_i) + M$ then M depends only on $|C|$ (Lemma 7) 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 $w < M$.

Theorem 5 (Appendix). *Given states $s, t \in C$ the set $G(C, s, t) = \{(x, y, n_1, \ldots, n_k) \mid (s, x) \rightarrow^* (t, y)$ in C^γ where $\gamma(p_i) = n_i\}$ is $\exists$PAD definable.*

Recall that satisfiability of $\exists$PAD formulae is in NP [15] and that $|C|$ is exponential in $|A|$ (Lemmas 2 and 4). Hence, Theorem 3 follows. We have also proved the corresponding lower bound, in fact, already for a single parameter.

Theorem 6 (Appendix). *The halting problem for parametric timed automata with one parametric clock and one parameter is NEXP-hard.*

The proof of $\exists$PAD definability relied on two lemmas that we prove now. First, we show how to calculate the odd-indexed runs. Let $\boldsymbol{c_1}, \boldsymbol{c_2}$ be configurations of C^γ between two successive parameters: $\gamma(p_i) < counter(\boldsymbol{c_1}), counter(\boldsymbol{c_2}) < \gamma(p_{i+1})$.

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 $\geq p_j$ and $\leq p_k$ for $k \leq i < j$. And all transitions $\leq p_j$ and $\geq p_k$ for $k \leq i < j$ are replaced by $+0$ transitions. Further, 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.

Consider some run $\pi : \boldsymbol{c_1} \rightarrow \boldsymbol{c_2}$ 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 [13]: 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 7 ([13], Lemma 42). *Let C_i be as above. There is a constant M (polynomial in $|C_i|$) s.t. for any configurations $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ of C_i the following holds: let $U = \min(counter(\boldsymbol{c_1}), counter(\boldsymbol{c_2}))$ and $V = \max(counter(\boldsymbol{c_1}), counter(\boldsymbol{c_2}))$. If $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ then there is a run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ such that $U - M \leq counter(\pi) \leq V + M$.*

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

Lemma 8. *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 $\gamma(p_i) + M < counter(\boldsymbol{c}), counter(\boldsymbol{d}) < \gamma(p_{i+1}) - M$. Then*

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

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 $\gamma(p_i) - M \leq counter(\pi) \leq \gamma(p_i) + M$. Then for every component $\pi(i)$, we can write $counter(\pi(j)) = \gamma(p_i) + z_j$ 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. (See the appendix)

Lemma 9. *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*

$$((s, \gamma(p_i) + x), (t, \gamma(p_i) + y)) \in C^\gamma(\gamma(p_i) - M, \gamma(p_i) + M) \iff$$
$$((s, \delta(p_i) + x), (t, \delta(p_i) + y)) \in C^\delta(\delta(p_i) - M, \delta(p_i) + M)$$

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.

4 Two Parametric Clocks

We show that the halting problem for parametric timed automata with two parametric clocks is equivalent to the halting problem for parametric bounded one-counter machines. The equivalence is used in Section 4.2 to show decidability of the halting problem in certain cases.

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

Theorem 10. *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 halting problem. Moreover, if C has no '$\equiv 0 \bmod c$' transitions then A has no nonparametric clocks. Otherwise, A has one nonparametric clock.*

4.1 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. To begin, we construct (Lemma 2) a parametric 0/1 timed automaton B with two parametrically constrained clocks, denoted x and y, with the halting problem equivalent to A. We then 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 12 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, so 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. Precisely, suppose we know the order of the parameters $p_1 < p_2 < \ldots < p_k$. Then 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.[3]

Therefore, machine C guesses 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) and it also guesses 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 in which the next reset occurs and which clock is reset next (see Fig 3).

Machine C checks that the sequence is valid as follows. First, C checks, that $(z, 0)$ lies in R_0. Second, it checks that the regions are adjacent: $i_{l+1} - i_l = 1 \wedge j_{l+1} = j_l$ or $i_{l+1} = i_l \wedge j_{l+1} - j_l = 1$ 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. Then, C checks that starting in clock configuration $(z, 0)$, the regions can be visited in the guessed order.

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 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 Fig. 3.

- 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. In Fig. 3 this corresponds to region $R_{(1,0)}$, which is visited before $R_{(2,0)}$.
- Similarly, if $j_{l+1} - j_l = 1$. E.g, in Fig. 3 region $R_{(2,1)}$ is visited before $R_{(2,2)}$.

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

We say that R_t was ***reached from left*** in the first and that R_t was ***reached from bottom*** in the second case. See Fig. 3 for the intuition behind the names.

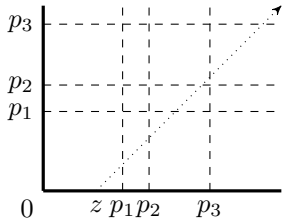

Fig. 3. Regions for three parameters $p_1 < p_2 < p_3$. The dotted line shows an en evolution of clock configuration, which visits $R_{(0,0)}, R_{(1,0)}, R_{(2,0)}, R_{(2,1)}, R_{(2,2)}, R_{(3,2)}, R_{(3,3)}$.

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. Each $B_{(i,j)}$ corresponds to the region $R_{(i,j)}$ in the same way automata C_i corresponded to one-dimensional regions in Section 3.

Notice that $B_{(i,j)}$'s are 0/1 automata without resets or comparisons, 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: $\Pi(X, s, t) = \{v \mid \exists \pi \in X.\, start(\pi) = (s, 0) \wedge end(\pi) = (t, v)\}$.

Lemma 11 (Appendix). *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: $\Pi(X, s, t) = C \cup \bigcup_{1 \le j \le r} \{a_j + b_j \mathbb{N}\}$ where $C \subseteq \mathbb{N}$ is finite and $a_j, b_j \in \mathbb{N}$.*

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 mundane and is presented in full in the appendix. 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. Fig. 4).

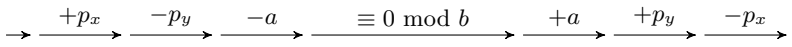

Fig. 4. Gadget testing that for given $a, b \in \mathbb{N}$ there is $k \in \mathbb{N}$ such that $z + p_x - p_y = a + kb$,i.e., $z + p_x - p_y - a \equiv 0 \bmod b$. Letter z denotes the current counter value.

Finally note 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 $[0, 2p_k]$). 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 12. 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.

Theorem 13. *For a parametric timed automaton A with two parametric clocks there is a parametric bounded one-counter machine C with equivalent halting problem.*

Our reduction was inspired by the work in [8] (see the Related Work section). 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 [8].

4.2 The One-Parameter Case

Suppose that the parametric two-clock timed automaton A uses only a single parameter p. Consider the corresponding counter machine C. It turns out that, C has no '$+[0, p]$' transitions, all '$\equiv 0 \bmod c$' transitions can be eliminated from C and if there is an accepting run in C then there is one which is bounded (Remark 12). This yields a decidable class of counter machines (Lemma 15).

Consider C. Inspecting the detailed proof of the reduction (as found in the Appendix), observe that '$+[0, p]$' transitions are introduced only when two successive regions are visited from the same direction (both from left or both from bottom). For a single parameter, only four regions exist $[0, p] \times [0, p], [0, p] \times [p, \infty], [p, \infty] \times [0, p], [p, \infty] \times [p, \infty]$. Simple case analysis shows that this can happen only when the counter starts at 0, which can be treated separately.

So C has no '$+[0, p]$' transitions. Next, we also eliminate '$\equiv 0 \bmod c$' transitions from C. Let $K = \{c_1, \ldots, c_r\}$ 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 $D = (d_1, \ldots, d_r)$, let C_D be the 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 $S \times \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$ where S are the states of C and $\mathbb{Z}_{c_i}$ denotes the ring of integers modulo c_i. The machine C_D contains all comparison transitions of C. Further, let $(v_1, \ldots, v_r) \in \mathbb{Z}_{c_1} \times \ldots \times \mathbb{Z}_{c_r}$. Then C_D also contains the following transitions:

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

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 14. *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 $((s, x \bmod c_1, \ldots, x \bmod c_r), x) \to^* ((t, y \bmod c_1, \ldots, y \bmod c_r), y)$ in C_D^γ.*

Suppose we guess D then, by Remark 12, to decide the halting problem in C_D it suffices to find an accepting run π such that $counter(\pi) \leq 2 \cdot \gamma(p)$. Now, for any such run π and index i we can write $counter(\pi(i)) = a\gamma(p) + b$ where $a \leq 2$ and $b < \gamma(p)$. Since a is bounded, we can 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 7 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 for every τ_i the value $counter(\tau_i) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. And for every π_i we have $counter(\pi_i) \bmod \gamma(p) \in (M, \gamma(p) - M)$. Then we use techniques on factoring of runs analogous to those used for one parametric clock (Section 3.1). In general, we have: (See the appendix.)

Lemma 15. *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 $G(C,s,t,k) = \{(x,y,q) \mid \exists \pi : (s,x) \to (t,y) \in C^\gamma$ s.t. $counter(\pi) \leq k \cdot q$ where $q = \gamma(p)\}$ is $\exists$PAD definable.*

Theorem 16. *The halting problem is decidable for parametric timed automata with two parametric clocks and a single parameter.*

This settles the case of parametric timed automata with two clocks and a single parameter. The case of arbitrary many parameters, or even two parameters, is left as an open problem. However, already for a single parameter, we establish the following lower bound. (See the appendix.)

Theorem 17. *The halting problem for parametric timed automata with two parametric clocks and a single parameter is $PSPACE^{NEXP}$-hard.*

Acknowledgments. This research was financially supported by EPSRC.

References

1. R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the 25th Annual Symposium on Theory of Computing*. ACM Press, 1993.
2. 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.
3. L. Doyen. Robust parametric reachability for timed automata. *Information Processing Letters*, 102(5):208 – 213, 2007.
4. 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.

5. 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.
6. C. Haase. *On the Complexity of Model Checking Counter Automata*. PhD thesis, University of Oxford, 2012.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. L. Lipshitz. The Diophantine Problem for Addition and Divisibility. *Transactions of The American Mathematical Society*, 235, 1978.
15. L. Lipshitz. Some remarks on the diophantine problem for addition and divisibility. volume 33, 1981.
16. 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.

A Proof of Lemma 4

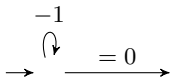

Fig. 5. Gadget implementing a clock reset.

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. Each transition resetting a clock is then replaced by a gadget (see Fig. 5) that subtracts 1 from the counter until the counter equals 0. □

B Proof of Theorem 5

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

Now, by Lemma 8, the odd-indexed runs π_{2i+1} correspond to runs in some $C_{h(2i+1)}$, which by Lemma 1 are ∃PAD expressible as:

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

where $h(2i+1)$ denotes the appropriate one-counter machine $C_{h(2i+1)}$ the run π_{2i+1} lies in. By taking a conjunction of the corresponding ∃PAD formulae we obtain a single ∃PAD formula.

Precisely, given state $s_1, \ldots, s_l$ 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 9) 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). The last conjunct encodes the restriction $\gamma(p_i) + M < \gamma(p_{i+1})$ from Lemmas 8 and 9.

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 than 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. Hence, considering the set of all such formulae we conclude that the reachability relation in C is $\exists$PAD definable. □

C Proof of Lemma 8

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 $(\gamma(p_i), \gamma(p_{i+1}))$:

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

It thus suffices to show that

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

The left-to-right implication is immediate. For the converse, suppose that $\boldsymbol{c} \rightarrow^* \boldsymbol{d}$ in C_i^{γ}. Since, $\gamma(p_i) + M < \mathit{counter}(\boldsymbol{c}), \mathit{counter}(\boldsymbol{d}) < \gamma(p_{i+1}) - M$, by Lemma 7, there exists a run π in C_i^{γ} from $\boldsymbol{c}$ to $\boldsymbol{d}$ such that $\gamma(p_i) < \mathit{counter}(\pi) < \gamma(p_{i+1})$. Hence $(\boldsymbol{c}, \boldsymbol{d}) \in C_i^{\gamma}(\gamma(p_i), \gamma(p_{i+1}))$ as required. □

D Proof of Lemma 9

Proof. For any run π from $(s, \gamma(p_i) + x)$ to $(t, \gamma(p_i) + y))$ in C^{γ} such that $\gamma(p_i) - M < \mathit{counter}(\pi) < \gamma(p_i) + M$ and for any index j we can write $\mathit{counter}(\pi(j)) = \gamma(p_i) + z_j$ 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. 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 7, 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 $(t, \gamma(p_i) + z')$. □

E Proof of Theorem 6

Proof. The proof is by reduction from succinct SAT, a well-known NEXP-complete problem. A similar construction was used in [5] to show that LTL model checking of parametric one-counter machines is NEXP-hard.

The succinct SAT problem is the adaptation of the classical SAT problem in which the input formula is not given explicitly but instead the input consists of a Boolean circuit $\mathbb{C}$ encoding the formula.

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

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

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 $v : \{x_1, x_2, \ldots\} \to \{0, 1\}$ is encoded in p as follows. Let p_k be the k-th prime. Then if $p_k | p$ then the variable x_k in φ is set to true in the assignment: $v(x_k) = 1$. And if $p_k \not| p$ then x_k is set to false in the assignment: $v(x_k) = 0$. 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.

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 Fig 6). Hence, φ is satisfiable if and only 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 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 and so many different ways exist of using timed automata to calculate PSPACE functions. A particular approach is sketched below.

```
for b ← 0 to 1^n do                                   ▷ Try all possible assignments
  s ← false                                           ▷ Denotes whether some literal evaluates to true
  for i ← 0 to 2 do                                   ▷ Try all three literals
    v · w ← ℂ(b · i)
    x ← 𝕊_{n^th prime}(v)                             ▷ Calculate the corresponding prime number
    if w = 0 ∧ 𝕊_{divides}(p, x) then
      s ← true
    end if
    if w = 1 ∧ 𝕊_{notdivides}(p, x) then
      s ← 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
```

Fig. 6. Algorithm iterating over all clauses of φ and checking that each is satisfiable. $\mathbb{S}_{divides}(p, x)$ stands for subroutine calculating that x divides p and $x \leftarrow \mathbb{S}_{n^{th}prime}(v)$ assigns the nth prime to x.

Since A has no restrictions on the number of nonparametric clocks, we use two 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[4]. So 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, 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 Fig. 6, 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 gadgets shown in Fig. 7 to check (non)divisibility of p by the prime numbers. Notice that the gadgets and hence the entire proof uses only a single parameter p. □

F Proof of Theorem 10

Proof. A similar proof appeared in [8]. 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.

[4] Alternatively, resetting the 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

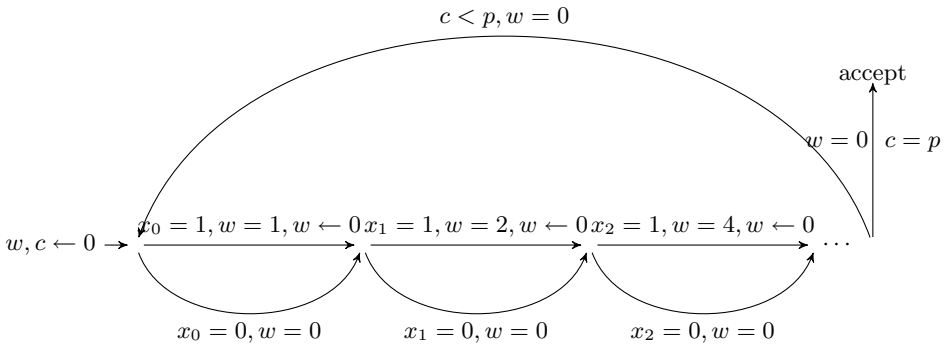

Fig. 7. PTA implementing x divides p. The bits $x_0, x_1, x_2, \ldots, x_m$ represent $x = \sum_i x_i 2^i$ in binary. An auxiliary clock w is used to count to the powers of 2. The gadget is entered on the left and one traversal through the gadget takes x time units. Clock c measures p. Note that the transition to the accepting state can be taken only if x divides p. Gadget checking $x \not| p$ is obtained by changing the final test $w = 0, c = p$ to $w = 0, c > p$

Timed automaton A has the same set of parameters as C together with a fresh parameter M. Now, whenever clock x or y reaches M the clock is reset. Intuitively, M is an upper bound on counter values in an accepting run of C. Further, counting modulo 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. 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. So z counts modulo c and finally we just need to check that when $y = M$ then $z = c$.

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. □

G Proof of Lemma 11

Proof. It was shown in [6], Lemma 4.1.18, that the reachability relation in non-parametric one-counter machines is definable in existential fragment of Presburger arithmetic (without divisibility). It is well known that this fragment defines effectively semilinear sets. □

H Reduction from Parametric Timed Automata with Two Parametrically Constrained Clocks

We give here missing details on how to check that c_{l+1} is reachable from c_l in region B_l and how to simulate clock resets.

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 $p_{x+1} - p_x \in \Pi(s_l, s_{l+1})$, which can be checked by C (similarly as in Fig. 4).
- 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 $p_{y+1} + z - p_x \in \Pi(s_l, s_{l+1})$, which can be checked by C (cf. Fig. 4).
- 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 $p_{x+1} - p_y - z \in \Pi(s_l, s_{l+1})$, 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 $p_{y+1} + z - p_y - z = p_{y+1} - p_y \in \Pi(s_l, s_{l+1})$, which can be checked by C.

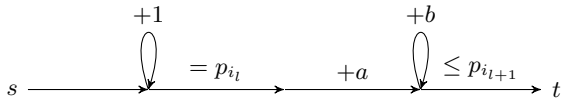

Fig. 8. A gadget implementing a reset of clock y in the left/left situation.

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

By modifying B if necessary, we can assume that all edges resetting a clock are $+0$ edges without any comparison. 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 clock y.

Simulation of reset distinguishes whether R_l and R_{l+1} were reached from bottom or from left. By Lemma 11, 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 the 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 $p_x + \Pi(s_l, t) \in [p_x, p_{x+1}]$. 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 Fig. 8.
- 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 $z' - p_x - a \equiv 0 \bmod b$.
- 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 $z \bmod b$ using a '$\equiv 0 \bmod b$' transitions, then incrementing the counter using $+[0, p_{y-1} - p_y]$ and then checking that the new counter value z' satisfies $z' \equiv z + a \bmod b$.

This finishes a reset of y. For the reset of x, we proceed analogously. Consider the four cases:

- R_l **reached from left,** R_{l+1} **reached from left**. Thus, B reaches R_l in configuration $(s_l, (p_x, p_x - z))$ and resets when clock $y < p_y - z$. Recall that after reset, the new counter value z' stores $-y$. Thus, $z' \in [z - p_{x+1}, z - p_x]$. Analogously to (bottom, bottom) case above, B subtracts p_x from the counter and then B subtracts a number of the form $a + kb \in [0, p_{x+1} - p_x]$.
- R_l **reached from left,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in configuration $(s_l, (p_x, p_x - z))$ and resets when $y < p_{y+1}$.
  So, $z' \in [-p_{y+1}, z - p_x]$. So B subtracts p_y and then a from the counter and then B keeps subtracting b checking that the counter is above $-p_{x+1}$.
- R_l **reached from bottom,** R_{l+1} **reached from left**. Thus, B reaches R_l in configuration $(s_l, (p_y + z, p_y))$ and resets when clock $y < p_{x+1} - z$.
  So, $z' \in [z - p_{x+1}, -p_y]$. Hence, B subtracts p_{x+1} and then adds a to the counter and then B keeps adding b checking that the counter is below $-p_y$.
- R_l **reached from bottom,** R_{l+1} **reached from bottom**. Thus, B reaches R_l in configuration $(s_l, (p_y + z, p_y))$ and resets when $y < p_{y+1}$. So, $z' \in [-p_{y+1}, -p_y]$. So B sets the counter to $p_y - a$ and then it keeps subtracting b checking that the counter is above $-p_{y+1}$.

This finishes the simulation of a single stage of B.

I Proof of Lemma 15

Proof. Let S be the set of states of C. We build a one-counter machine G with states $S \times \{0, \ldots, k-1\}$. Intuitively, if $z < \gamma(p)$, the configuration $((s,i),z)$ of G represents the configuration $(s, i\gamma(p)+z)$ of C. Machine G 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 7, there is $M \in \mathbb{N}$ such that we can assume that runs deviate by at most M from the initial and the final counter values.

Let $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$ be two configurations of C. Even without the knowledge of $\gamma(p)$, we show that there is a run from $\boldsymbol{c_1}$ to $\boldsymbol{c_2}$ that splits into finitely many subruns of a simple form. (The proof is analogous to the proof of Theorem 5.)

Consider a shortest run $\pi : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C^γ. We 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 for every τ_i the value $\mathit{counter}(\tau_i) \bmod \gamma(p) \in [0, \ldots, M] \cup [\gamma(p) - M, \gamma(p))$. And for every π_i we have $\mathit{counter}(\pi_i) \bmod \gamma(p) \in (M, \gamma(p) - M)$. For a run ρ and a subset $S \subseteq \mathbb{N}$ of natural numbers, the notation $\mathit{counter}(\rho) \in S$ denotes the fact that for every i we have $\mathit{counter}(\rho(i)) \in S$.

Notice that the configuration $\mathit{start}(\tau_i)$ is determined by the state of C, $\lfloor \mathit{counter}(\mathit{start}(\tau_i))/\gamma(p) \rfloor$ and $\mathit{counter}(\mathit{start}(\tau_i)) \bmod \gamma(p)$. Now, C has only finitely many states, $\lfloor \mathit{counter}(\mathit{start}(\tau_i))/\gamma(p) \rfloor \leq k$ and $\mathit{counter}(\mathit{start}(\tau_i)) \bmod \gamma(p)$ can have only one of $2M+1$ values. Thus, τ_i can start in only $O(nkM)$ possible configurations. 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 $l = O(nkM)$.

The $\exists$PAD definability of the existence of such a split is analogous to the proof for machines with only $-1, 0$ and $+1$ updates (Theorem 5). As in Lemma 8, the existence of π_i can be witnessed by G. Precisely, for configurations $\boldsymbol{c_1}, \boldsymbol{c_2}$ of C^γ such that $\mathit{counter}(\boldsymbol{c_1}) \bmod \gamma(p), \mathit{counter}(\boldsymbol{c_2}) \bmod \gamma(p) \in (M, \gamma(p) - M)$ we have $\boldsymbol{c_1} \to^* \boldsymbol{c_2}$ in G if and only if there is a run $\pi' : \boldsymbol{c_1} \to \boldsymbol{c_2}$ in C^γ such that $\mathit{counter}(\pi') \bmod \gamma(p) \in (M, \gamma(p) - M)$.

Further, as in Lemma 9, the existence of runs τ_i is independent of $\gamma(p)$ and can be precomputed[5]. Now, the runs in G are $\exists$PAD expressible (Lemma 1). By

[5] 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 mimick the transitions in G. Now, the graph is independent of $\gamma(p)$ and so we can calculate rechability between any pair of vertices. This gives τ_i's.

taking a conjunction of the corresponding ∃PAD formulae we obtain a single ∃PAD formula

$$\left(\bigwedge_i Reach(D, start(\pi_i), end(\pi_i))(counter(start(\pi_i)), counter(end(\pi_i)), p)\right) \wedge \psi$$

defining the reachability relation for a particular factoring where as in Theorem 5, 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 ∃PAD formulae, we conclude that the reachability relation is ∃PAD definable. □

J Proof of Theorem 16

Proof. Consider the corresponding counter machine C. By Remark 12, we can assume that the counter of C remains in $[0, 2p]$. Lemma 15 above shows that reachability up to any fixed constant multiple of p is ∃PAD definable.

So, the decision procedure guesses $D \in \mathbb{Z}_{c_1} \ldots \times \mathbb{Z}_{c_r}$ and then checks the satisfiability of $G(C_D, s_0, f, 2)(0, y, p) \wedge \bigwedge_i (\gamma(i) \equiv d_i \bmod c_i)$ where s_0 is the initial state and f is a final state of C_D. The expression $x \equiv y \bmod c_i$ is expressible using a ∃PAD formula $\varphi_i(x, y)$ equal to $\exists q \,.\, (x = c_i \cdot q + y) \ \wedge \ (y < c_i)$. □

K Proof of Theorem 17

Proof. We first describe the PSPACE$^{\text{NEXP}}$-hard problem we reduce from. Recall from the proof of Lemma 6 that Succinct SAT, whereby a 3SAT formula is represented by a circuit, is a NEXP-complete problem. So consider a PSPACE Turing Machine T that makes NEXP oracle calls by means of Succinct SAT. Precisely, T is allowed to generate polynomial-size circuits encoding 3SAT formulae and then pass them to a Succinct SAT oracle. It is clear that such machines can solve PSPACE$^{\text{NEXP}}$ 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 Succinct SAT. This automaton is then used inside a nonparametric timed automaton as a NEXP oracle. Hence, the PSPACE$^{\text{NEXP}}$-hardness follows.

The construction of A is an extension of ideas from Lemma 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 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 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. It 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 we outlined how A can calculate and represent π_k in binary $\pi_k = \sum_i v_i 2^i$ where v_i are bits.

Then checking whether π_k divides the assignment is done by modifying the gadget from Fig. 7 used for the divisibility in Lemma 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 k. The algorithm that is hard-wired into A is shown in 9.

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 Fig. 9 to take an encoding of a circuit and accept if and only if the corresponding formula is satisfiable. Since Succinct SAT is NEXP-complete, the algorithm can be used as a general NEXP oracle. The resulting automaton is constructible in polynomial

```
for i ← 1 to 2(3 · 2^n)^2 do
  if 𝕊_notdivides(p, i) then                ▷ Checks that lcm(1, ..., 2(3 · 2^n)^2) | p
    reject
  end if
end for
ass ← 0                                     ▷ Stores the value of the assignment
while ass ≠ p do                            ▷ Try all possible assignments
  for b ← 0 to 2^n do                       ▷ Iterate over all clauses
    ok ← true                               ▷ Denotes whether the current assignment satisfies φ
    s ← false
    for i ← 0 to 2 do                       ▷ Check if some literal is true under the current assignment
      v · w ← ℂ(b · i)
      x ← 𝕊_{n^th prime}(v)
      if w = 0 ∧ 𝕊_divides(p, x) then
        s ← true
      end if
      if w = 1 ∧ 𝕊_notdivides(p, x) then
        s ← true
      end if
    end for
    if s = false then
      ok ← false
    end if
  end for
  if ok = true then                         ▷ Accept if found a satisfying assignment
    accept
  end if
end while
reject                                      ▷ No assignment satisfies φ; reject
```

Fig. 9. 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.

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 PSPACE^{NEXP} problems to the halting 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. □