Verification of Timed Asynchronous Programs

Parosh Aziz Abdulla
Uppsala University, Sweden
parosh@it.uu.se

Mohamed Faouzi Atig
Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

Shankara Narayanan Krishna
IIT Bombay
krishnas@cse.iitb.ac.in

Shaan Vaidya
IIT Bombay
shaan@cse.iitb.ac.in

Abstract

In this paper, we address the verification problem for timed asynchronous programs. We associate to each task, a deadline for its execution. We first show that the control state reachability problem for such class of systems is decidable while the configuration reachability problem is undecidable. Then, we consider the subclass of timed asynchronous programs where tasks are always being executed from the same state. For this subclass, we show that the control state reachability problem is PSPACE-complete. Furthermore, we show the decidability for the configuration reachability problem for the subclass.

2012 ACM Subject Classification Theory of Computation

Keywords and phrases Reachability, Timed Automata, Asynchronous programs

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2018.8

1 Introduction

One of the well-known design paradigms in concurrent programs is to break a problem into smaller subproblems which are solved asynchronously and concurrently. Each process or thread in the program can then dispatch tasks to other processes, expecting them to be completed by a certain deadline. Each process has a potentially unbounded bag where its pending tasks are stored. In the asynchronous paradigm, one need not wait for time-consuming tasks to be completed to proceed; asynchronous procedure calls are stored in a task buffer, which are executed later, rather than right away. The tasks which are posted asynchronously have deadlines attached to them, and the process or thread, in whose bag the task has been posted, must execute the task within the deadline. In addition to asynchronous procedure calls, one can also make use of synchronous procedure calls where the caller of the procedure blocks until the callee returns. To summarize, an asynchronous program is one that contains procedure calls which are not immediately executed from the calling site, but stored and dispatched in a non-deterministic order by some scheduler(s) at a later point.

As an example for timed asynchronous programs, we look at *SwingWorker*, an abstract class developed for the Swing library of Java, and is used to perform lengthy GUI interaction tasks in a background thread. While developing applications, sometimes the GUI hangs when it is trying to do some lengthy task. For such purposes, the *SwingWorker* class

schedules the execution of this lengthy task on a different thread while the GUI still remains responsive. There are deadlines associated with the background tasks, and if the worker thread which is handling the background task does not finish by the given deadline, then an interrupt is created. To update the user (and GUI) regarding the progress of background tasks, inter-thread communication is allowed.

Writing correct asynchronous programs and reasoning about their correctness is very difficult, since the creation and execution of tasks within deadlines leads to unpredictable behaviours. The verification of asynchronous programs is hence a very challenging topic. A formal model of mutiset pushdown systems for asynchronous recursive programs was presented in [16]. This model consists of a pushdown automaton equipped with a multiset or bag. The automaton adds pending asynchronous method calls to the bag, and the stack executes synchronous recursive method calls. A task can be taken from the bag for execution when the stack is empty. The control state reachability problem was shown to be decidable with an EXPSPACE lower bound under this model. This shows that the case of single-thread asynchronous programs, the reachability problem is very difficult. Subsequently, [8] showed that control state reachability for single-thread asynchronous recursive programs is EXPSPACE-complete. In all these models, time constraints do not play a role in the execution of the asynchronous methods. In the timed setting, [7] considers asynchronous calls of the form `future(p,t)` posted to the task buffer, where p is a handler and $t \in \mathbb{N}$. The idea is that the handler p will execute the task in t time units from now. The execution of the program is controlled by logical ticks of a clock. The model proposed in [7] is a generalization of the models in [16] and [11]. [7] shows that safety checking for such programs is undecidable.

The goal of this paper is to investigate the decidability and complexity of the reachability problem for asynchronous non-recursive programs under dense time. We propose a formalism called multiset timed automata (MTA) where each process is modeled as a timed automaton [2]. Each timed automaton is equipped with a bag or multiset. To handle asynchronous method calls, each timed automaton can post a task to the bag of another automaton. These tasks have deadlines attached to them. The deadline is either a natural number $d \in \mathbb{N}$ or ∞. When a task is posted to a bag, its age is considered to be 0, and with elapse of time, the age also grows. A task can be executed by the process in whose bag it lies, before the age of the task exceeds the deadline; tasks whose ages have exceeded the deadline will be forever pending. While a main process picks up pending tasks depending on their ages in [7], in our model, a process can execute a pending task in its bag at its will. There are 2 sources of infinity in our model: one coming from dense-time, and the second coming from the unbounded size of the bags of each process. We investigate control state reachability as well as configuration reachability of this model, and show that control state reachability is decidable and EXPSPACE-hard, while configuration reachability is undecidable. We then identify a practically relevant class of MTA where the task execution happens from the same state in each process, and give a PSPACE-complete decision procedure for control state reachability. The configuration reachability also turns out to be decidable for this class.

Related Work

Most of the existing work (e.g., [3, 4, 6, 8, 11, 13, 14, 16]) on the formal verification of asynchronous programs considers the untimed version. In [7], the authors consider timed constraints on tasks; however, this model is different from the formal model studied in this paper. In fact, in [7], the authors assume that a task should always be executed by its deadline and the execution of each task is done in logical zero time. In our model, a task whose age has exceeded the deadline will be forever pending. Furthermore, the control

state reachability for the model presented in [7] is undecidable while it is decidable for our model. In [5], the authors consider a similar model than the one considered in this paper and show that the coverability problem is decidable using a different technique than ours.

An extended version of this paper is available at [1].

2 Preliminaries

In this section, we introduce some notations and definitions that will be used throughout the paper.

Notations

We use standard notation $\mathbb{N}$ for the set of naturals, along with ∞. $\mathbb{R}$ represents the set of non-negative real numbers. Let $\mathcal{X}$ be a finite set of variables called *clocks*, taking values from $\mathbb{R}$. A *valuation* on $\mathcal{X}$ is a function $\nu : \mathcal{X} \to \mathbb{R}$. We assume an arbitrary but fixed ordering on the clocks and write x_i for the i-th clock. This allows us to treat a valuation ν as a vector $(\nu(x_1), \nu(x_2), \ldots, \nu(x_n))$ in $\mathbb{R}^{|\mathcal{X}|}$. For a subset of clocks $X \in 2^{\mathcal{X}}$ and valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, we write $\nu[X{:=}0]$ for the valuation where $\nu[X{:=}0](x) = 0$ if $x \in X$, and $\nu[X{:=}0](x) = \nu(x)$ otherwise. For $t \in \mathbb{R}$, write $\nu + t$ for the valuation defined by $\nu(x) + t$ for all $x \in \mathcal{X}$. The valuation $\mathbf{0} \in \mathbb{R}^{|\mathcal{X}|}$ is a special valuation such that $\mathbf{0}(x) = 0$ for all $x \in \mathcal{X}$. For $a, b \in \mathbb{N}$ and $a < b$, the set $\mathcal{I}$ of time intervals is defined by $\mathcal{I} := [a, b] \mid [a, a] \mid (a, b] \mid [a, b) \mid (a, b) \mid [a, \infty) \mid (a, \infty)$. The set of clock constraints, denoted $\varphi(\mathcal{X})$, is the set of Boolean formulae over $\{x \in I \mid x \in \mathcal{X}, I \in \mathcal{I}\}$. For a constraint $g \in \varphi(\mathcal{X})$, and a valuation $\nu \in \mathbb{R}^{|\mathcal{X}|}$, we write $\nu \models g$ to represent the fact that valuation ν satisfies the constraint g. For example, $(1.1, 0, 10) \models (x_1 \in (0, 2)) \wedge (x_2 \in [0, 0]) \wedge (x_3 \in (1, \infty))$.

Timed Automata

Let *Act* denote a finite set called actions. A timed automaton (TA) [2] is a tuple $\mathcal{A} = (L, L^0, Act, \mathcal{X}, E)$ such that (i) L is a finite set of locations, (ii) $\mathcal{X}$ is a finite set of clocks, (iii) *Act* is a finite alphabet called an action set, (iv) $E \subseteq L \times \varphi(\mathcal{X}) \times Act \times 2^{\mathcal{X}} \times L$ is a finite set of transitions, and (v) $L^0 \subseteq L$ is the set of initial locations. A state s of a timed automaton is a pair $s = (\ell, \nu) \in L \times \mathbb{R}^{|\mathcal{X}|}$. A time elapse transition from $s = (\ell, \nu)$ to $s' = (\ell', \nu')$ denoted $s \xrightarrow{t} s'$ is defined iff $\ell' = \ell$ and $\nu' = \nu + t$. Given $e = (\ell, g, a, Y, \ell') \in E$, a discrete transition from s to s' on e is written as $s \xrightarrow{e} s'$, such that $\nu \models g$ and $\nu' = \nu[Y{:=}0]$. A run is a finite sequence $\rho = s_0 \xrightarrow{t_1} s'_0 \xrightarrow{e_1} s_1 \xrightarrow{t_2} s'_1 \xrightarrow{e_2} s_2 \ldots s_{n-1} \xrightarrow{t_n} s'_{n-1} \xrightarrow{e_n} s_n$ of states with alternating time elapse transitions and discrete transitions.

Multisets or Bags

A *multiset* or *bag* over an alphabet Σ is a mapping $M : \Sigma \mapsto \mathbb{N}$. For an element $a \in \Sigma$, we use $a \in M$ to denote that $M(a) \geq 1$. We use $\emptyset$ to denote the empty multiset. Given two multisets M_1, M_2 over Σ, we write $M_1 \leq M_2$ iff $M_1(a) \leq M_2(a)$ for all $a \in \Sigma$. $M_1 + M_2$ denotes the multiset M such that $M(a) = M_1(a) + M_2(a)$ for all $a \in \Sigma$. Likewise, $M_1 - M_2$ denotes, when it is defined (i.e., $M_1 \geq M_2$), the multiset M such that $M(a) = M_1(a) - M_2(a)$ for all $a \in \Sigma$. The notation $M_1 + a$ denotes a multiset M_2 such that

$M_2(a) = M_1(a) + 1$ and $M_2(b) = M_1(b)$ for all $b \neq a$. Likewise, $M_1 - a$ denotes, when it is defined (i.e. $M_1(a) \geq 1$), a multiset M_2 such that $M_2(a) = M_1(a) - 1$ and $M_2(b) = M_1(b)$ for all $b \neq a$. The terms multiset and bag will be used interchangeably.

Timed Petri Nets

A Timed Petri Net (TPN) [17] is a tuple $\mathcal{N} = (P, T, F, c)$ where P is a finite set of places, T is a finite set of transitions, $T \cap P = \emptyset$ and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, $c : F \cap (P \times T) \to \mathcal{I}$ is a time constraint relation assigning a time interval to every arc from a place to a transition. A marking M of $\mathcal{N}$ is a mapping that associates to each place p a multiset over $\mathbb{R}$. A marked TPN is a pair $(\mathcal{N}, M_0)$ where M_0 is an initial marking, which assigns to each place in P, an initial multiset of tokens annotated with 0 (the initial age). The dynamics of a TPN consists of two types of transitions rules: firing of a transition and time elapsing. Given $\mathcal{N}$, along with a marking M, (denoted $(\mathcal{N}, M)$) a transition t is enabled at M iff for all places p such that $(p, t) \in F$, there exists some $x \in M(p)$, and $x \in c(p, t)$. If t is enabled by M, then it can be fired, producing a marking M' obtained from M by (i) removing a token from $M(p)$ for all places p such that $(p, t) \in F$ and whose age satisfies $c(p, t)$, and (ii) adding a token with age 0 to $M(q)$ for all places q such that $(t, q) \in F$. In a time elapse transition, with an elapsing time $r \in \mathbb{R}$, the age of all tokens increases by r. A marked TPN $(\mathcal{N}, M_0)$ induces a transition system with states are the markings of $\mathcal{N}$, and the transition relation consists of time elapsing and firing transitions.

A read arc in a TPN facilitates firing a transition without removing the token. We use $F* \subseteq P \times T$ to denote the set of read arcs and $c* : F* \to \mathcal{I}$ to denote a function that assigns a time interval to each read arc. A transition t is enabled iff for all places p such that $(p, t)* \in F*$, there exists some $x \in M(p)$ and $x \in c*(p, t)$. The transition system induced by a marked TPN with read arcs can be defined in a similar manner as for marked TPN. A 1-safe marking is one where $|M(p)| \leq 1$ for all $p \in P$. A 1-safe TPN is a marked TPN $(\mathcal{N}, M_0)$, with $F \cap F* = \emptyset$, where all markings which are reachable from M_0 are 1-safe.

Coverability problem. For markings M_1 and M_2 in a TPN $\mathcal{N}$, define $M_1 \leq M_2$ iff for all $p \in P$, $M_1(p) \leq M_2(p)$. The coverability problem for $\mathcal{N}$ asks whether, given a marking M, it is possible to reach a marking M' in $\mathcal{N}$ from the initial marking M_0 such that $M \leq M'$.

3 Multiset Timed Automata

Let $\mathcal{T}=\{T_1, \ldots, T_N\}$ be a set consisting of $N \geq 1$ timed automata $T_i=(L_i, L_i^0, Act_i, \mathcal{X}_i, E_i)$. A Multiset Timed Automata (MTA) is defined as $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$, where Σ is a finite alphabet called *tasks*, $\mathcal{X} = \biguplus_{i=1}^{N} \mathcal{X}_i$ is the finite disjoint union of clocks in T_i, St is a function that assigns a finite multiset $St(i)$ over Σ (possibly empty) to the timed automaton T_i. This is the initial set of tasks assigned to T_i. The actions Act_i are defined as $Act_i = \{i!j(a[d]), i?a \mid a \in \Sigma, j \in \{1, \ldots, N\}, d \in \mathbb{N} \cup \{\infty\}\} \cup \{\mathsf{nop}_i\}$. The number d is the *deadline* for the task a. The action $i!j(a[d])$ represents T_i adding the task a to the bag of automaton T_j, and the task a has an associated deadline d. Likewise, the action $i?a$ represents automaton T_i picking up the task a from its bag, provided its age has not exceeded the deadline. For readability reasons, we assume that any outgoing transition from any initial location is labeled by an action of the form $i?a$. We use the notation N-MTA whenever we need to clarify the number of timed automata T_i which are used in the definition.

Let $\overline{q} = (q_1, \ldots, q_N)$ be a tuple of states, where $q_i = (s_i, \nu_i)$ is the current state of T_i. Let $\overline{m} = (M_1, \ldots, M_N)$ be a tuple of multisets. Each element in M_i has the form

$\alpha = (a, r, d) \in \Sigma \times \mathbb{R} \times \mathbb{N}$ consisting of – *pending tasks*, their *ages*, and their *deadlines* in T_i. The age of a task in a bag is the time elapse since it has been added to the bag. For $t \in \mathbb{R}$, let $\overline{q} + t$ represent the tuple $(q'_1, \ldots, q'_n)$ where $q'_i = (s_i, \nu_i + t)$. For an element $\alpha = (a, r, d) \in M_i$, $\alpha + t = (a, r + t, d)$; $M_i + t$ is the multiset obtained by replacing each $\alpha \in M_i$ with $\alpha + t$. We define $\overline{m} + t$ as the tuple $(M_1 + t, \ldots, M_N + t)$.

A configuration $\mathfrak{c}$ of an N-MTA is the tuple $(\overline{q}, \overline{m})$ consisting of the current states of all the N timed automata, along with the multisets of pending tasks corresponding to each T_i. An initial configuration is defined as $\mathfrak{c}_0 = (\overline{q}_0, \overline{m}_0)$, where $\overline{q}_0$ is the tuple $((\ell^0_1, \mathbf{0}), \ldots, (\ell^0_N, \mathbf{0}))$ of initial states of all T_i ($\ell^0_i \in L^0_i$) and $\overline{m}_0 = (M_1, \ldots, M_N)$ where $M_i((a, r, d)) = St(i)(a)$, for all $a \in \Sigma$, $r = 0$ and $d = \infty$, and $M_i((a, r, d)) = 0$ otherwise. Given two configurations $\mathfrak{c} = (\overline{q}, \overline{m})$, and $\mathfrak{c}' = (\overline{q'}, \overline{m'})$, we have:

- For $t \in \mathbb{R}$, $\mathfrak{c} \xrightarrow{t} \mathfrak{c}'$ is a time elapse transition iff $\overline{q'} = \overline{q} + t, \overline{m'} = \overline{m} + t$.
- Let $e_i = (\ell_i, g_i, act_i, Y_i, \ell'_i) \in E_i$. Then, $\mathfrak{c} \xrightarrow{e_i} \mathfrak{c}'$ iff
  - $q_i = (\ell_i, \nu_i)$, $\nu_i \models g_i$, $q'_i = (\ell'_i, \nu'_i)$, $\nu'_i = \nu_i[Y_i := 0]$, and for all $k \neq i$, $q'_k = q_k$, and,
  - If $act_i = i!j(a[d])$, then $M'_j = M_j + (a, 0, d)$, and $M'_k = M_k$ for all $k \neq j$,
  - If $act_i = i?a$, then $\exists c, d$, such that $(a, c, d) \in M_i$, $M'_i = M_i - (a, c, d)$, and $c \leq d$ (i.e. the age of the task has not yet exceeded the deadline) and $M'_k = M_k$ for all $k \neq i$,
  - If $act_i = \mathsf{nop}_i$, then $M'_k = M_k$ for all $1 \leq k \leq N$.

Starting with an initial configuration $\mathfrak{c}_0$, a run ρ is defined as a finite sequence of alternating time elapse and discrete transitions of the form $\mathfrak{c}_0 \xrightarrow{t_0} \mathfrak{c}'_0 \xrightarrow{e_1} \mathfrak{c}_1 \xrightarrow{t_1} \mathfrak{c}'_1 \xrightarrow{e_2} \mathfrak{c}_2 \cdots \xrightarrow{e_j} \mathfrak{c}_j$ or $\mathfrak{c}_0 \xrightarrow{t_0} \mathfrak{c}'_0 \xrightarrow{e_1} \mathfrak{c}_1 \xrightarrow{t_1} \mathfrak{c}'_1 \xrightarrow{e_2} \mathfrak{c}_2 \cdots \xrightarrow{t_j} \mathfrak{c}'_j$. In that case we say the configuration $\mathfrak{c}_j$ is reachable from the initial configuration $\mathfrak{c}_0$ by the run ρ.

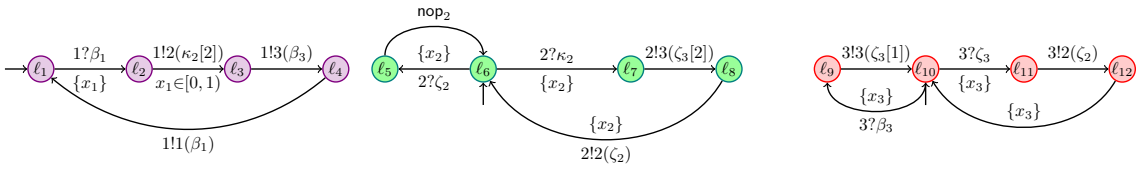

Figure 1 A stateless and time independent 3-MTA consisting of timed automata T_1, T_2, T_3 from left to right. When the deadline of a task is ∞, we do not mention it.

In this paper, we consider the following problems. Let $\overline{s} = (s_1, \ldots, s_N) \in L_1 \times \cdots \times L_N$.

P1 **Control State Reachability**. Given a particular tuple of locations $\overline{s} = (s_1, \ldots, s_N)$ of an N-MTA $\mathcal{M}$, the control state reachability problem asks if starting from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$, there is a run reaching a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ such that $q_i = (s_i, \nu_i)$ for some $\overline{m}$, and for some ν_i, for all $1 \leq i \leq N$.

P2 **Configuration Reachability**. Given a particular tuple of locations $\overline{s} = (s_1, \ldots, s_N)$ of an N-MTA $\mathcal{M}$, the configuration reachability problem asks if starting from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$, there is a run reaching a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ such that $\overline{m} = (\emptyset, \ldots, \emptyset)$ and $q_i = (s_i, \mathbf{0})$, for all $1 \leq i \leq N$.

Stateless and Time-Independent MTA

An N-MTA is said to be *stateless* if $E_i \cap (L_i \setminus \{\ell_i^0\} \times \varphi(\mathcal{X}_i) \times \{i?a | a \in \Sigma\} \times 2^{\mathcal{X}_i} \times L_i) = \emptyset$ for all $1 \leq i \leq N$, and some $\ell_0^i \in L_0^i$. The stateless condition ensures that a new task can be picked by an automaton only from a unique initial location. An N-MTA is said to be *time-independent* iff, in each T_i, all clocks are reset on picking a task from the multiset, and no clock constraints are checked (i.e. $E_i \cap (L_i \times \varphi(\mathcal{X}_i) \times \{i?a | a \in \Sigma\} \times (2^{\mathcal{X}_i} \setminus \{\mathcal{X}_i\}) \times L_i) = \emptyset$ and $E_i \cap (L_i \times (\varphi(\mathcal{X}_i) \setminus \{true\}) \times \{i?a | a \in \Sigma\} \times 2^{\mathcal{X}_i} \times L_i) = \emptyset$ for all $1 \leq i \leq N$).

Figure 1 describes a stateless and time-independent MTA $\mathcal{M}$ consisting of 3 timed automata T_1, T_2, T_3. The following is a run in $\mathcal{M}$. The initial configuration $\mathfrak{c}_0 = (\overline{q}_0, \overline{m}_0)$ where $\overline{q}_0 = ((\ell_1, 0), (\ell_6, 0), (\ell_{10}, 0))$ and $\overline{m}_0 = (M_1, M_2, M_3)$ with multisets $M_1 = \{(\beta_1, 0, \infty)\}$, $M_2 = \{(\beta_2, 0, \infty)\}$, and $M_3 = \{(\beta_3, 0, \infty)\}$. Let $e_{i,j}$ denote the transition from location ℓ_i to ℓ_j (in the example, we have at most one transition between any pair of locations ℓ_i, ℓ_j). For example, $e_{23} = (\ell_2, x_1 \in [0,1), 1!2(\kappa_2[2]), \emptyset, \ell_3)$. Consider the run σ $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2}} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3}} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7}} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8}} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{e_{10,9}} \mathfrak{c}_5 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4}} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1}} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2}} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6}} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{e_{9,10}} \mathfrak{c}_{10} \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5}} \mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11}} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12}} \mathfrak{c}_{13}$ which reaches locations $(\ell_2, \ell_5, \ell_{12})$ in T_1, T_2, T_3 respectively.

4 Control State Reachability

In the following, we first prove that the control reachability is decidable with a non-primitive complexity (at the level F_{ω^ω} in the fast growing hierarchy [9]). Then, we show that the control state reachability for stateless and time independent MTA is PSPACE-complete.

▶ **Theorem 1.** *The control state reachability problem for N-MTA is reducible to the coverability problem for timed Petri nets with read-arcs.*

Proof. We give a translation from an N-MTA $\mathcal{M}$ to a TPN with read arcs $\mathcal{N}$ such that the control state reachability of $\mathcal{M}$ reduces to the coverability of $\mathcal{N}$.

Let $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$ be an N-MTA. Without loss of generality, assume that we are interested in reaching $\overline{f} = (f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$. Given the N-MTA $\mathcal{M}$, we construct a timed Petri net $\mathcal{N}$ as follows. There is a place p_ℓ in the net corresponding to each location $\ell \in L_i$ in T_i for each $i \in \{1, \ldots, N\}$. For each T_i, there is one and only one marked place p_ℓ such that $\ell \in L_i$, to denote that the control of T_i is at a certain location ℓ. For each clock x in $\mathcal{X}$, we have a place p_x in the net. Next, we model the multisets M_i of each T_i. Let $d_{max} \in \mathbb{N}$ be the maximal value used for any deadline in $\mathcal{M}$. The possible task, deadline combinations are in the set $\Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$. Therefore, corresponding to each T_i, we have $|\Sigma| \times (d_{max} + 2)$ places in the net. We need to have these many places so as to distinguish between the tokens. Thus, for each pair $(a, d) \in \Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$, we have the places $p^1_{(a,d)}, \ldots, p^N_{(a,d)}$.

A transition of the form $(\ell, g, i?a, Y, \ell')$ in automaton T_i is simulated by a transition in $\mathcal{N}$ as follows. A token from the place p_ℓ corresponding to the location ℓ, and a token from one of the places $p^i_{(a,d)}, d \in \{0, 1, \ldots, d_{max}, \infty\}$ are removed. The deadline is checked on the arc via a constraint $[0, z]$ from the place $p^i_{(a,z)}$ containing the token. A token is added to the place $p_{\ell'}$ corresponding to ℓ'. A transition of the form $(\ell, g, i!j(a[d]), Y, \ell')$ in automaton T_i is simulated in a similar way. The tokens corresponding to locations are removed and added as in the previous case and a token is added to the place $p^j_{(a,d)}$. The clock constraints corresponding to any transition are checked using read arcs from the places simulating the clocks. Clock resets are simulated by removing a token and putting back a token in the place corresponding to the clock.

The details of the formal construction of $\mathcal{N}$ and the correctness proof can be found in the extended version of the paper [1].

◀

As a corollary of Theorem 1, we get:

▶ **Corollary 2.** *The control state reachability problem for (time-independent) N-MTA is decidable.*

Observe that we can easily show that the coverability of Petri nets is reducible to the control state reachability problem for time-independent (resp. stateless) N-MTA (in the same way as the proof of EXPSPACE lower bound for the model multiset pushdown systems presented in [16]). Therefore, the control state reachability problem for time-independent (resp. stateless) N-MTA is EXPSPACE-hard.

In the rest of this section, we consider the case of stateless and time-independent N-MTA.

▶ **Theorem 3.** *The control state reachability problem for stateless and time-independent N-MTA is PSPACE-complete (for $N \geq 1$).*

Proof. Since MTA subsume timed automata [2], the PSPACE-hardness of the control state reachability of MTA follows directly from the PSPACE-hardness of reachability of timed automata. The rest of the proof is devoted to proving the PSPACE-membership of the problem.

Let $\mathcal{M} = (\Sigma, \mathcal{T}, \mathcal{X}, St)$ be a stateless, time-independent N-MTA, with $\mathcal{T} = \{T_1, \ldots, T_N\}$ $\mathcal{X} = \biguplus_{i=1}^{N} \mathcal{X}_i$ and St, the function that assigns an initial multiset $St(i)$ to each timed automaton T_i. Incurring a polynomial blowup in the size, we give a reduction from the control state reachability of $\mathcal{M}$ to the coverability in 1-safe timed Petri net with read arcs. The coverability of 1-safe timed Petri nets with read arcs is known to be PSPACE-complete [17] and our result follows from this.

Without loss of generality, assume that we are interested in reaching $\overline{f} = (f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$. Let σ be any run from the initial configuration $\mathfrak{c}_0$ of $\mathcal{M}$ which leads into a configuration with locations $\overline{f}$. Let $\mathfrak{c} = (\overline{q}, \overline{m})$ be any configuration that appears in σ. Our proof is divided into two parts.

1. We show that the number of *relevant* task tuples along σ is bounded by N. Intuitively, A task tuple $(a, r, d) \in \Sigma \times \mathbb{R} \times \mathbb{N}$ is relevant for an automaton T_i if $(a, r, d) \in M_j$, for some j, $(r \leq d)$ and the task (a, r, d) must be executed by T_j in order to reach the location f_i. The irrelevant task tuples can hence be ignored from each M_i, as they do not affect the control state reachability.
2. The bound on the number of relevant task tuples obtained in the previous step is used in constructing a reachability preserving 1-safe timed Petri net with read arcs.

Bounding the number of relevant task tuples

Consider the run σ as described above. Starting from $\mathfrak{c}_0$, let σ_i denote the sequence of transitions (in the order they appear in σ), pertaining only to T_i. In the run σ pertaining to the example in figure 1, σ_1 consists of all the violet discrete transitions separated by time elapses: $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2}} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3}} \mathfrak{c}_2 \xrightarrow{1.1} \mathfrak{c}'_5 \xrightarrow{e_{3,4}} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1}} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2}} \mathfrak{c}_8$. We now define a block.

A *block* in σ_i begins with a discrete transition of the form $\xrightarrow{i?a}$ (for some task a) and extends until the next transition of the form $\xrightarrow{i?b}$ (for some task b) is encountered. Thus, a

block is a sequence of transitions between two executions of tasks by some T_i, and begins with some task execution. σ_1 has two blocks: the sequence of transitions from $\mathfrak{c}'_0$ till $\mathfrak{c}'_7$ forms a block, and the second block is the transition from $\mathfrak{c}'_7$ to $\mathfrak{c}_8$. Omitting the time elapse transitions in σ_i, let us label each transition in σ_i with a unique name. Doing this for all σ_i gives us a unique label for each discrete transition in σ. Let $\mathcal{L} = \{\alpha_1, \ldots, \alpha_m\}$ be the set of block labels occurring in σ. In our running example, using labels $\{\alpha_1, \ldots, \alpha_6\}$, we can label the blocks in σ as $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2},\alpha_1} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3},\alpha_1} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7},\alpha_2} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8},\alpha_2} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{e_{10,9},\alpha_3} \mathfrak{c}_5 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4},\alpha_1} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1},\alpha_1} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2},\alpha_4} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6},\alpha_2} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{e_{9,10},\alpha_3} \mathfrak{c}_{10} \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5},\alpha_5} \mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11},\alpha_6} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12},\alpha_6} \mathfrak{c}_{13}$. From here on, we refer to the blocks using the block labels.

For each timed automaton T_i, we now analyze the blocks which contribute in reaching the desired location f_i. The last block α of σ_i, which contains the last task tuple (a, r, d) executed by T_i definitely contributes to T_i reaching f_i. Likewise, the block α' which added this last task a to the bag of T_i also contributes to T_i reaching f_i (note that block α' may start with a task b which is executed by T_j, $j \neq i$). We can continue backwards in this manner and say that the block α'' which added the task b to the bag of T_j also contributes to T_i reaching f_i and so on. Given a block label α, let $\mathsf{dep}(\alpha)$ denote the set of timed automata T_i such that α contributes to T_i reaching f_i. Thus, if α is the last block in T_i, then $i \in \mathsf{dep}(\alpha)$ (we just write the indices i rather than T_i). Likewise, if $i \in \mathsf{dep}(\alpha)$ and if α' is the block which added the task a which was executed at the beginning of α, then $i \in \mathsf{dep}(\alpha')$, and so on. $\mathsf{dep}(\alpha)$ is called the dependency set of α. In our running example above, $3 \in \mathsf{dep}(\alpha_6)$ since α_6 is the last block for T_3; however the task ζ_3 which was executed in block α_6 was added in block α_2 ($e_{7,8}$), and the task κ_2 executed in block α_2 was added in block α_1. Thus, $3 \in \mathsf{dep}(\alpha_6), \mathsf{dep}(\alpha_2), \mathsf{dep}(\alpha_1)$.

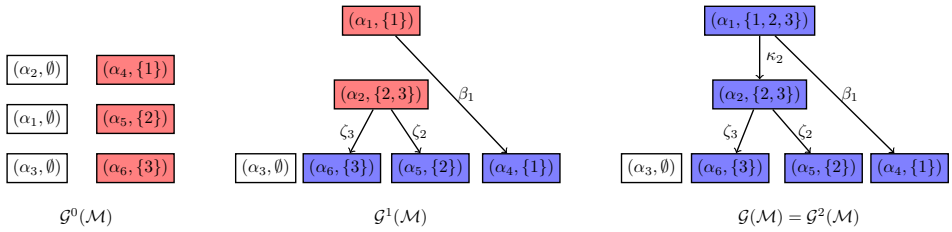

Figure 2 The dependency graph in stages. $\mathcal{G}^0(\mathcal{M})$ is the initial graph with no edges. $\mathcal{G}^{i+1}(\mathcal{M})$ is obtained from $\mathcal{G}^i(\mathcal{M})$ by changing the color of all the red vertices in $\mathcal{G}^i(\mathcal{M})$. The graph stabilizes when there are no red vertices.

We construct a *dependency graph* $\mathcal{G}(\mathcal{M})$ which keeps track of the dependencies between blocks. Define a function $g : \mathcal{L} \to (\Sigma \times \{1, \ldots, N\} \times \mathcal{L}) \cup \{\perp\}$ which maps a block label α to the triple (a, i, α') if block α begins with $(i?a)$ the execution of task a, which was added to the bag of T_i by block α'. If a is part of the initial multiset ($a \in St(i)$) then $g(\alpha) = \perp$. The vertex set of $\mathcal{G}(\mathcal{M})$ is the set of pairs $(\alpha, \mathsf{dep}(\alpha))$ where α is a block label and $\mathsf{dep}(\alpha)$ is its dependency set. $\mathcal{G}(\mathcal{M})$ is a graph with colored vertices, and is built inductively. To begin, there are no edges, and we have the following vertices.

- Vertices $(\alpha, \{i\})$ and α is the last block of T_i. To begin, we are sure of $i \in \mathsf{dep}(\alpha)$. We color these vertices red.
- Vertices $(\alpha, \emptyset)$, and α is not the last block for any T_i. To begin, we have not yet discovered whether α contributes to any T_i, so we keep $\mathsf{dep}(\alpha) = \emptyset$. The information with respect

to $\mathsf{dep}(\alpha)$ will be updated when we discover that α contributed to some T_i. We color these vertices white.

To add the edges, we repeat the following procedure until no red vertices remain. In each step, we choose a red vertex $(\alpha, \mathsf{dep}(\alpha))$ and do the following.

1. If $g(\alpha) = \bot$, then color $(\alpha, \mathsf{dep}(\alpha))$ blue,
2. If $g(\alpha) = (a, i, \alpha')$ and $(\alpha', \mathsf{dep}(\alpha'))$ is white, then color $(\alpha, \mathsf{dep}(\alpha))$ blue and color $(\alpha', \mathsf{dep}(\alpha'))$ red. Update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$, and add an edge $\xrightarrow{a}$ from $(\alpha', \mathsf{dep}(\alpha'))$ to $(\alpha, \mathsf{dep}(\alpha))$.
3. If $g(\alpha) = (a, i, \alpha')$ and $(\alpha', \mathsf{dep}(\alpha'))$ is not white, then color $(\alpha, \mathsf{dep}(\alpha))$ blue, update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$, and add an edge $\xrightarrow{a}$ from $(\alpha', \mathsf{dep}(\alpha'))$ to $(\alpha, \mathsf{dep}(\alpha))$.

Finally, we update the dependency relation dep of the vertices as follows: If $(\alpha, \mathsf{dep}(\alpha))$ and $(\alpha', \mathsf{dep}(\alpha'))$ are blue with $g(\alpha) = (a, i, \alpha')$, then update $\mathsf{dep}(\alpha')$ to be $\mathsf{dep}(\alpha') \cup \mathsf{dep}(\alpha)$. Note that the above procedure terminates, since the number of blue vertices in each step increases. The final graph obtained as result is $\mathcal{G}(\mathcal{M})$. Figure 2 describes constructing $\mathcal{G}(\mathcal{M})$ for the run ρ discussed above. Consider any vertex $(\alpha, \mathsf{dep}(\alpha))$ colored blue in $\mathcal{G}(\mathcal{M})$. Clearly, this vertex contributes to all T_i such that $i \in \mathsf{dep}(\alpha)$. Consider any path in $\mathcal{G}(\mathcal{M})$ from a vertex with no incoming edges to a vertex with no outgoing edges. There is at least one such path since the last task executed along σ corresponds to the last block of some T_i which has not contributed to any T_j. A path $v_1 \dots v_s$ in $\mathcal{G}(\mathcal{M})$ is a dependency path for automaton T_i if the vertex $v_s = (\alpha, \mathsf{dep}(\alpha))$, and α is the last block for T_i. Let us go back to our running example run σ using Figure 2. The tasks appearing on the edges of $\mathcal{G}(\mathcal{M})$ are the relevant tasks. From $\mathcal{G}(\mathcal{M})$, the relevant task in the bags when the second block α_2 started is κ_2. κ_2 is executed at the beginning of block α_2. Relevant tasks ζ_2, ζ_3 are added to the bag in block α_2, and α_1 adds β_1. β_1 is executed in block α_4 while ζ_2, ζ_3 respectively are executed in blocks α_5, α_6. The relevant tasks along run σ are $\beta_1, \kappa_2, \zeta_2, \zeta_3$, of which at most 3 are stored across bags at any point of time. Thus, we can obtain another run σ' which is reachability equivalent to σ as follows. The block α_3 is useless as it is not contributing to any of the automata. Each block begins at a unique initial location of some automaton, and, on the transition which executes the task, it does not check any constraints, and resets all clocks on the transition. Due to this, we can "prune away" a block from a run, and reconnect the run at a later block if we maintain the time elapse in the interim. Hence, removing a useless block of some automaton T_i does not affect the control reachability, since the next useful block of T_i again starts from the same initial location of T_i. Accounting for the time elapse in the useless block is sufficient to ensure that the ages of the pending tasks are accurate. σ' can be constructed with the rest of the blocks, using only the relevant tasks: $\mathfrak{c}_0 \xrightarrow{0.5} \mathfrak{c}'_0 \xrightarrow{e_{1,2},\alpha_1} \mathfrak{c}_1 \xrightarrow{0.3} \mathfrak{c}'_1 \xrightarrow{e_{2,3},\alpha_1} \mathfrak{c}_2 \xrightarrow{0.5} \mathfrak{c}'_2 \xrightarrow{e_{6,7},\alpha_2} \mathfrak{c}_3 \xrightarrow{0.2} \mathfrak{c}'_3 \xrightarrow{e_{7,8},\alpha_2} \mathfrak{c}_4 \xrightarrow{0} \mathfrak{c}'_4 \xrightarrow{0.4} \mathfrak{c}'_5 \xrightarrow{e_{3,4},\alpha_1} \mathfrak{c}_6 \xrightarrow{0.1} \mathfrak{c}'_6 \xrightarrow{e_{4,1},\alpha_1} \mathfrak{c}_7 \xrightarrow{0.1} \mathfrak{c}'_7 \xrightarrow{e_{1,2},\alpha_4} \mathfrak{c}_8 \xrightarrow{0.1} \mathfrak{c}'_8 \xrightarrow{e_{8,6},\alpha_2} \mathfrak{c}_9 \xrightarrow{0.2} \mathfrak{c}'_9 \xrightarrow{0.6} \mathfrak{c}'_{10} \xrightarrow{e_{6,5},\alpha_5} \mathfrak{c}_{11} \xrightarrow{0.5} \mathfrak{c}'_{11} \xrightarrow{e_{10,11},\alpha_6} \mathfrak{c}_{12} \xrightarrow{0.9} \mathfrak{c}'_{12} \xrightarrow{e_{11,12},\alpha_6} \mathfrak{c}_{13}$.

We want to prove that in any configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ appearing in the run σ, the number of pending tasks maintained in $\overline{m} = (M_1, \dots, M_N)$ which contribute, in reaching the desired control states, in σ is $\leq N$. These are the relevant tasks, and each one is part of a block α, and the corresponding vertex $(\alpha, \mathsf{dep}(\alpha))$ in $\mathcal{G}(\mathcal{M})$ is colored blue. If we attach the color of the vertex $(\alpha, \mathsf{dep}(\alpha))$ to the task a in $g(\alpha)$, then we want to prove that in any configuration appearing in σ, the number of blue tasks is $\leq N$. Assume that there is some configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ in σ such that the number of blue tasks in $\overline{m}$ is $p > N$. Let $a_1, \dots, a_p$ be the tasks in $\overline{m}$, and let $\alpha_1, \dots, \alpha_p$ be the blocks where these are executed. Since $p > N$, and there are only N multisets in $\overline{m}$, there are at least two tasks a_i, a_j such that $\mathsf{dep}(\alpha_i) \cap \mathsf{dep}(\alpha_j) \neq \emptyset$.

Observe that, by definition, we have $\mathsf{dep}(\alpha_k) \neq \emptyset$ for all $k \in \{1, \ldots, p\}$. Let us assume that $k \in \mathsf{dep}(\alpha_i) \cap \mathsf{dep}(\alpha_j)$. Since both are blue, both get executed in σ, and both lie in the dependency path of the last block of the automaton T_k. Clearly, one must come before the other, and the earlier block has contributed to the creation of the later block. Hence, they cannot be pending at the same time. Thus, the number of blue pending tasks in any configuration is bounded above by N.

Construction of 1-safe TPN with read arcs

Now, we are ready to propose a 1-safe timed Petri net (with read-arcs) whose coverability problem is equivalent to the control state reachability problem of the given N-MTA.

Given the N-MTA $\mathcal{M}$ consisting of timed automata $T_1, \ldots, T_N$, we construct a 1-safe TPN $\mathcal{N}$. There is a place p_ℓ corresponding to each location $\ell \in L_i$ in T_i. For each T_i, there is one and only one marked place p_ℓ at any point in the execution, such that $\ell \in L_i$, to denote that the control of T_i is at a certain location ℓ. For each clock x in X, there is a place p_x. Next, we model the multisets M_i of each T_i. Let $d_{max} \in \mathbb{N}$ be the maximal value used for any deadline in $\mathcal{M}$. For each task $a \in \Sigma$, we have $|\Sigma| \times (2 + d_{max})$ possible combinations of tasks and associated deadlines. The bound established above tells us that there are at most N pending tasks in any configuration i.e. at any point we will have to keep track of N tasks but they can be distributed in any of the multisets. There are $|\Sigma| \times (d_{max} + 2)$ possibilities for task, deadline pairs. Tasks will be modeled as tokens in the net. So to be able to distinguish between them, for each T_i, we need $N \times |\Sigma| \times (d_{max} + 2)$ places (N, because 1-safe). For each T_i and for each pair $(a, d) \in \Sigma \times \{0, 1, \ldots, d_{max}, \infty\}$, we have N places $p^i_{(a,d,1)}, \ldots, p^i_{(a,d,N)}$.

A transition of the form $(\ell^0_i, g, i?a, Y, \ell')$ in automaton T_i is simulated by $N \times (d_{max} + 2)$ transitions in $\mathcal{N}$ as follows. For each $(z, j) \in \{0, 1, \ldots, d_{max}, \infty\} \times \{1, \ldots, N\}$, a transition removes a token from the place $p_{\ell^0_i}$ corresponding to the unique initial location ℓ^0_i, a token from $p^i_{(a,z,j)}$ and adds a token to the place $p_{\ell'}$ corresponding to ℓ'. The deadline is checked on the arc from the place $p^i_{(a,z,j)}$ by a constraint which checks the age of the token to be in the interval $[0, z]$. As any deadline value is possible, and any of the N places can be filled, one of the $N \times (d_{max} + 2)$ transitions is non-deterministically chosen.

A transition of the form $(\ell, g, i!j(a[d]), Y, \ell')$ in automaton T_i is simulated in a similar way by $N + 1$ transitions. In each of the N of these transitions, tokens for control locations are added and removed as in the previous case. For each $k \in \{1, \ldots, N\}$, one of the N transitions adds a token to the place $p^j_{a,d,k}$ if it is empty. The $(N+1)$-th transition simulates the possibility that the task a is not relevant (only N are relevant at any point) and so it simulates only the change in control location and adds no other tokens. One of these $N + 1$ transitions is chosen non-deterministically. Observe that the first N transitions add a token only to an empty place $p^j_{a,d,k}$ by definition of an 1-safe Petri net.

Clock resets are simulated by adding and removing a token from the corresponding place p_x for the clock. Clock constraints are simulated by read arcs. These arcs are connected with the corresponding transitions that are described above.

The formal construction is in [1]. Thus, the control state reachability in $\mathcal{M}$ to reach $(f_1, \ldots, f_N) \in L_1 \times \cdots \times L_N$ reduces to the coverability problem of the marking M given by $M(p_{f_i}) = 1$ for all $1 \leq i \leq N$ (and hence $M(p_\ell) = 0$ for all $\ell \notin \{f_1 \ldots, f_N\}$). The control state reachability of $\mathcal{M}$ thus reduces to the coverability of the constructed 1-safe timed Petri net with read arcs. Since the coverability of 1-safe timed Petri nets with read arcs is PSPACE-complete [17], the control state reachability of $\mathcal{M}$ is also PSPACE-complete. ◀

5 Configuration Reachability

In this section, we explore the general question of the configuration reachability problem for N-MTA. We first show (theorem 4) that the configuration reachability problem for N-MTA is undecidable.

▶ **Theorem 4.** *The configuration reachability problem for N-MTA is undecidable. This undecidability holds even in the case of time-independent N-MTA.*

Proof. The proof is done by a reduction from the reachability problem for a 2-counter machine (which is known to be undecidable [15]). The main idea is to construct an 1-MTA whose set of states contains the states of the two counter machine plus some auxiliary states that are used to simulate the zero tests as we will see later on. The 1-MTA has two types of tasks a and b. The number of pending tasks of type a (resp. b) corresponds to the value of the counter c_1 (resp. c_2). Furthermore, the 1-MTA has one clock x that is used to check that no time elapsed when simulating some transitions of the two counter machine.

To simulate an increment transition of the form $(q, c_1{+}{+}, q')$ (resp.$(q, c_2{+}{+}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally adds a pending task of type a (resp. b) with zero as its deadline. Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a decrement transition of the form $(q, c_1{-}{-}, q')$ (resp. $(q, c_2{-}{-}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally consumes a pending task of type a (resp. b). Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a zero test transition of the form $(q, c_1 == 0, q')$ (resp. $(q, c_2 == 0, q')$) of the two counter machine, the 1-MTA proceeds as follows: (i) it checks that the value of the clock $x = 0$, (ii) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to one time unit, (iii) it will change its state from q to q', (iv) it checks that the value of the clock x is still zero, (v) it checks that one time unit has elapsed (ie., checking whether $x \in [1, 1]$) and resets the clock x, (vi) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to zero, and (vii) it checks that the value of $x = 0$. Here the auxiliary states are needed in the simulation of these steps.

Observe that if the 1-MTA reaches the final state with empty set of pending tasks, then all the simulation of the zero tests are performed correctly. Finally, note that the constructed 1-MTA is time independent. ◀

We now focus on the class of stateless and time-independent N-MTA.

▶ **Theorem 5.** *The configuration reachability problem for stateless and time-independent N-MTA is decidable.*

We begin by setting up some notations for the proof.

Well-quasi-orders and Higman's Lemma

Given a set $\mathcal{Q}$, a *quasi-order* on $\mathcal{Q}$ is a reflexive and transitive relation $\preceq \subseteq \mathcal{Q} \times \mathcal{Q}$. An infinite sequence $(q_1, q_2 \dots)$ in $\mathcal{Q}$ is said to be *saturating* if there exists indices $i < j$ such that $q_i \preceq q_j$. A quasi-order $\preceq$ is a *well-quasi-order* (*wqo*) [12] on $\mathcal{Q}$ if every infinite sequence

in $\mathcal{Q}$ is saturating. Let $\sqsubseteq$ be a quasi-order on $\mathcal{Q}$. The *induced monotone domination order* $\preceq$ on $\mathcal{Q}^*$, (i.e., the set of finite words over $\mathcal{Q}$) is defined as follows: $a_1a_2\ldots a_m \preceq b_1b_2\ldots b_n$ if there exists a strictly increasing function $g : \{1,2,\ldots,m\} \to \{1,2,\ldots,n\}$ such that, for all $1 \leq i \leq m$, $a_i \sqsubseteq b_{g(i)}$. It is well-known by *Higman's Lemma* [10] that if $\sqsubseteq$ is a wqo on $\mathcal{Q}$, then the induced domination order $\preceq$ is also a wqo on $\mathcal{Q}^*$. As an example, let $\Sigma = \{1,2,\ldots,12\}$ and let $\mathcal{Q}$ be the power set of Σ. Define $\sqsubseteq$ on $\mathcal{Q}$ to be the set inclusion relation. $\sqsubseteq$ is clearly a wqo since $\mathcal{Q}$ is finite. The induced monotone domination order $\preceq$ on $\mathcal{Q}^*$ is the subword order: for example, $\{1,2\}\{3\}\{5,6,7\} \preceq \{1,2,9\}\{1\}\{3,11\}\{12\}\{4,5,6,7\}$.

Encoding Configurations

We have seen in section 3 that a configuration of an N-MTA $\mathcal{M}$ is a tuple $(\overline{q}, \overline{m})$ where $\overline{q}$ is the sequence of states in each T_i, $1 \leq i \leq N$, and $\overline{m}$ is the tuple of multisets $(M_1,\ldots,M_N)$ corresponding to each T_i. Given $(\ell_1,\ldots,\ell_N) \in L_1 \times \cdots \times L_N$, we are interested in finding whether the configuration $\mathfrak{c}_{goal} = (\overline{q}, \overline{m})$ is reachable, where $\overline{q} = (q_1,\ldots,q_N)$, $q_i = (\ell_i, \mathbf{0})$ and $\overline{m} = (\emptyset,\ldots,\emptyset)$. A configuration $\mathfrak{c}$ is called *good* if $\mathfrak{c}_{goal}$ is reachable from $\mathfrak{c}$. A configuration is *bad* if it is not good. Clearly, $\mathfrak{c}_{goal}$ is reachable in $\mathcal{M}$ iff some initial configuration $\mathfrak{c}_0$ is good.

We now construct an equivalence relation on $\mathcal{M}$ by encoding the configurations of $\mathcal{M}$ as words over a certain alphabet. This will enable us to define a wqo on the resulting transition system. Let K be the maximal constant used in the clock constraints and deadlines in $\mathcal{M}$. Let $[K] = \{0,1,\ldots,K,\infty\}$. Let $\mathsf{reg} = \{r_0, r_1, \ldots, r_{2K}\}$ be a finite set of *regions*, where for $0 \leq i \leq K$, r_{2i} is defined as the singleton $\{i\}$, while r_{2i+1} is defined as the interval $(i, i+1)$ for $0 \leq i \leq K-1$. We also define the region r_{2K+1} as (K,∞). Let Γ_1 be the set $\mathcal{X} \times \mathsf{reg}$, and let Γ_2 be a multiset over $\{(a,r,j)_i \mid a \in \Sigma, r \in \mathsf{reg}, j \in [K], 1 \leq i \leq N\}$. Let Γ_3 be the set $\mathcal{X} \times r_{2K+1}$, and let Γ_4 be a multiset over $\{(a, r_{2K+1}, j)_i \mid a \in \Sigma, 1 \leq i \leq N, j \in [K]\}$.

Let Υ, Δ respectively be the power sets of $\Gamma_1 \cup \Gamma_2$ and $\Gamma_3 \cup \Gamma_4$. Let $\mathcal{L}{=}L_1 \times \cdots \times L_N$. We consider words of the form $\alpha w(P + \epsilon)$ where $\alpha \in \mathcal{L}$, $w \in \Upsilon^*$ and $P \in \Delta$. Since $\Upsilon, \Delta, \mathcal{L}$ are finite, they are all clearly well-quasi-ordered by set inclusion, and the set of words of the form $\alpha w(P+\epsilon)$ is well-quasi-ordered by the induced monotone domination order $\preceq$: $\alpha_1\rho_1\ldots\rho_m P_1 \preceq \alpha_2\gamma_1\ldots\gamma_n P_2$ if $\alpha_1 = \alpha_2$, $P_1 \subseteq P_2$, and there exists a strictly increasing function $g : \{1,2\ldots,m\} \to \{1,2,\ldots,n\}$ such that for all $1 \leq i \leq m$, $\rho_i \subseteq \gamma_{g(i)}$.

We next associate to any configuration $\mathfrak{c}$ of $\mathcal{M}$, a canonical word $\mathsf{W}(\mathfrak{c}) \in \mathcal{L} \cdot \Upsilon^* \cdot (\Delta + \epsilon)$. Let $y_{i,1},\ldots,y_{i,|\mathcal{X}_i|}$ be the set of clocks in T_i. Given a configuration $\mathfrak{c} = (\overline{q}, \overline{m})$ with $\overline{q} = ((\ell_1,\nu_1),\ldots,(\ell_N,\nu_N))$ and $\overline{m} = (M_1,\ldots,M_N)$, $\overline{q}$ is completely specified by describing for each $1 \leq i \leq N$, (i) the locations ℓ_i, (ii) the tuples $(\alpha_{i,j}, \mathsf{frac}(y_{i,j}))$ (resp, α_{ij}) if $\alpha_{ij}{=}((y_{i,j}, \mathsf{reg}(\nu(y_{i,j})))$ is in Γ_1 (resp. Γ_3) and $1 \leq j \leq |\mathcal{X}_i|$. Observe that here we use $\mathsf{frac}(y_{i,j})$ (resp. $\mathsf{reg}(\nu(y_{i,j}))$) to denote the fractional part (resp. the corresponding region) of $\nu(y_{i,j})$). The former case keeps track of clocks, their regions as well as the fractional parts of the clock valuations, while in the latter, the value of clock $y_{i,j}$ is more than K, (iii) the multi set consisting of tuples $(\beta_i, \mathsf{frac}(age(a)))$ (resp. β_i) if $\beta_i = (a, \mathsf{reg}(age(a)), d)$ is in Γ_2 (resp. Γ_4). The former keeps track of tasks, the region of their ages, and their deadlines, along with the fractional parts of the ages, while in the latter, the age of the task is more than K. Observe that $age(a)$ returns the age of the task a.

Next, we group together the symbols $\alpha_h \in \Gamma_1, \beta_g \in \Gamma_2$ having the same fractional parts. Notice that the fractional parts are retained only for clocks (tasks) whose value (age) has not yet exceeded K. This yields a new set of $\Gamma_1 \cup \Gamma_2$ letters paired with their fractional parts $\{(\zeta_i, \mathsf{frac}_i) \mid 1 \leq i \leq p\}$ where ζ_i is a (multi)set of symbols from $\Gamma_1 \cup \Gamma_2$ and frac_i is the fractional part of those symbols. p is the number of distinct fractional parts in $\mathfrak{c}$. We

then form the word $w = \rho_{i_{z_1}} \dots \rho_{i_{z_p}} \in \Upsilon^+$ where $z_1 \dots z_p$ is a permutation of $1 \dots p$ that puts $\mathsf{frac}_{z_1} \dots \mathsf{frac}_{z_p}$ in ascending order. Let $P \in \Delta$ be the set obtained (if any) by grouping all the symbols $\alpha_h \in \Gamma_3$ and $\beta_g \in \Gamma_4$. We then define $\mathsf{W}(\mathfrak{c}) = \alpha.w.P \in \mathcal{L}.\Upsilon^*(\Delta + \epsilon)$ as the canonical word encoding $\mathfrak{c}$.

▶ **Example 6.** Consider a 2-MTA $\mathcal{M}$. Let x_1, x_2 be the clocks of T_1 and y_1, y_2 be the clocks of T_2. Let $K = 3$ be the maximal constant used in $\mathcal{M}$. Consider the configurations
$\mathfrak{c}_1$=$((s_1, 0.5, 2.1), (s_2, 1.7, 2.5), (\{(a, 1.1, 2), (b, 2.3, \infty), (c, 3.5, \infty)\}, \{(d, 1.9, 2), (e, 0.7, 1)\}))$
and $\mathfrak{c}_2$=$((s_1, 0.5, 2.4), (s_2, 1.9, 2.5), (\{(a, 1.4, 2), (b, 2.45, \infty), (c, 3.9, \infty)\}, \{(d, 1.99, 2), (e, 0.9, 1)\}))$.
Then $\mathsf{W}(\mathfrak{c}_1) = \mathsf{W}(\mathfrak{c}_2) = \alpha w P$ where $\alpha = (s_1, s_2)$, $P = \{(c, r_7, \infty)_1\}$, and
$w = \{(x_2, r_5), (a, r_3, 2)_1\}\{(b, r_5, \infty)_1\}\{(x_1, r_1), (y_2, r_5)\}\{(y_1, r_3), (e, r_1, 1)_2\}\{(d, r_3, 2)_2\}$.

Two configurations $\mathfrak{c}_1, \mathfrak{c}_2$ are equivalent ($\mathfrak{c} \sim \mathfrak{c}'$) if $\mathsf{W}(\mathfrak{c}_1) = \mathsf{W}(\mathfrak{c}_2)$. A configuration $\mathfrak{c}_1$ is dominated by a configuration $\mathfrak{c}_2$ (written $\mathfrak{c}_1 \preceq \mathfrak{c}_2$) if writing $\mathfrak{c}_2 = (\overline{q}_2, \overline{m}_2)$, there exists $\overline{q}_1, \overline{m}_1$ such that and $\overline{m}_1 = (M'_1, \dots, M'_N)$ with $M'_i \subseteq M_i$ for all i, and $\mathfrak{c}_1 \sim (\overline{q}_1, \overline{m}_1)$. It can be easily seen that $\mathfrak{c}_1 \preceq \mathfrak{c}_2$ iff $\mathsf{W}(\mathfrak{c}_1) \preceq \mathsf{W}(\mathfrak{c}_2)$. In fact, the following lemma shows that $\sim$ is a bisimulation relation.

▶ **Lemma 7.** *Let $\mathfrak{c}_1, \mathfrak{c}_2$ be two configurations of an N-MTA. Let $e \in E_i$ be a transition, $1 \le i \le N$, and let $t \in \mathbb{R}$. If $\mathfrak{c}_1 \sim \mathfrak{c}_2$, then*
(1) If $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}'_1$, there exists $\mathfrak{c}'_2$ such that $\mathfrak{c}_2 \xrightarrow{e} \mathfrak{c}'_2$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$. If $\mathfrak{c}_2 \xrightarrow{e} \mathfrak{c}'_2$, there exists $\mathfrak{c}'_1$ such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}'_1$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$.
(2) If $\mathfrak{c}_1 \xrightarrow{t} \mathfrak{c}'_1$, there exists $\mathfrak{c}'_2$ and $t' \in \mathbb{R}$ such that $\mathfrak{c}_2 \xrightarrow{t'} \mathfrak{c}'_2$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$. If $\mathfrak{c}_2 \xrightarrow{t} \mathfrak{c}'_2$, there exists $\mathfrak{c}'_1$ and $t' \in \mathbb{R}$ such that $\mathfrak{c}_1 \xrightarrow{t'} \mathfrak{c}'_1$ and $\mathfrak{c}'_1 \sim \mathfrak{c}'_2$.

As an easy corollary of the above, we see that $\sim$ preserves goodness and badness: For any configurations $\mathfrak{c} \sim \mathfrak{c}'$, $\mathfrak{c}$ is good iff $\mathfrak{c}'$ is good. The proof follows from the definition of goodness and Lemma 7, whose proof can be found in the extended version of the paper [1].

It is hence sufficient to only consider configurations upto $\sim$-equivalence, and we define the quotient labeled transition system $\mathcal{M}/\sim$ to consist of all the words $\mathsf{W}(\mathfrak{c})$ whenever $\mathfrak{c}$ is a configuration of $\mathcal{M}$. Call $\mathcal{M}/\sim$ as $\mathcal{W}$. $\mathcal{W} = \{\mathsf{W}(\mathfrak{c}) \mid \mathfrak{c} \text{ is a configuration in } \mathcal{M}\}$. For $W_1, W_2 \in \mathcal{W}$, and a transition $e \in E_i$ for $1 \le i \le N$, we define a transition $W_1 \xrightarrow{e} W_2$ if for all $\mathfrak{c}_1 \in \mathsf{W}^{-1}(W_1)$, there is some configuration $\mathfrak{c}_2 \in \mathsf{W}^{-1}(W_2)$ such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}_2$. The timed transition is defined similarly. Corresponding to each initial configuration $\mathfrak{c}_0$ in $\mathcal{M}$, we consider $W_0 = \mathsf{W}(\mathfrak{c}_0)$ to be an initial word in $\mathcal{W}$. Let $\mathcal{W}_0$ be the set of initial words corresponding to initial configurations $\mathfrak{c}_0$. It can be seen that for any $W_1, W_2 \in \mathcal{W}$, and a transition $e \in E_i$ or $t \in \mathbb{R}$, $W_1 \xrightarrow{\alpha} W_2$ ($\alpha \in \{e, t\}$) iff there exist configurations $\mathfrak{c}_1 \in \mathsf{W}^{-1}(W_1)$ and $\mathfrak{c}_2 \in \mathsf{W}^{-1}(W_2)$ such that $\mathfrak{c}_1 \xrightarrow{\alpha} \mathfrak{c}_2$. Given a word $W \in \mathcal{W}$, and $\alpha \in \{e, t\}$ for some transition e and time $t \in \mathbb{R}$, let $\mathsf{succ}(W) = \{W' \in \mathcal{W} \mid W \xrightarrow{\alpha} W'\}$ denote the successors of W in $\mathcal{W}$.

▶ **Lemma 8.** *For any word W, the set $\mathsf{succ}(W)$ is finite and effectively computable.*

Let $\mathcal{W}_0$ be the set of initial words corresponding to $\mathsf{W}(\mathfrak{c}_0)$ for initial configurations $\mathfrak{c}_0$. Let $W_\emptyset = \mathsf{W}(\mathfrak{c}_{goal})$. Algorithm 1 decides whether the configuration $\mathfrak{c}_{goal}$ can be reached. In this algorithm, the function $\mathsf{Minimize}(R)$ is used, where $R \subseteq \mathcal{W}$ is a set of words. It does the following: it chooses a word $W_1 \in R$ and removes W_1 from R if there exists a word $W_2 \in R$ such that $W_2 \preceq W_1$, and then repeats the procedure until all words in R are processed. Overall, the algorithm works as follows. Till the set Next of words waiting to be processed is non-empty, the algorithm chooses one word from Next, and moves it to the $\mathsf{Processed}$ set. It also generates all successors of the chosen word, minimizes them, and adds them to Next

Algorithm 1 REACH EMPTY

Input: A stateless, time-independent N-MTA, and configuration $\mathfrak{c}_{goal} = (\overline{q}, \overline{m})$ as above.
Output: TRUE if $\mathfrak{c}_{goal}$ is reachable. Otherwise, FALSE.
if $W_\emptyset \in \mathcal{W}_0$, **then return** TRUE;
Processed $= \emptyset$;
Next $=$ Minimize($\mathcal{W}_0$);
while Next $\neq \emptyset$ **do**

1. Pick and remove a word W from Next and move it to Processed,
2. **foreach** $U \in$ Minimize(succ(W))
   a. **if** $U = W_\emptyset$, **then return** TRUE,
   b. **else if** $\nexists V \in$ Processed $\cup$ Next s.t. $V \preceq U$, **then**
   c. Remove all V from Processed $\cup$ Next s.t $U \preceq V$
   d. Add U to Next

return FALSE

unless there is already some $\preceq$-smaller word in Next or Processed. If a new word is added to Next, the algorithm removes at the same time all $\preceq$-bigger words from both Next and Processed. The correctness of the algorithm is discussed next.

A set of words R is good (denoted $Good(R)$) iff there exists some word $W \in R$ which is good. A word W is good iff there exists a good configuration $\mathfrak{c}$ such that $\mathsf{W}(\mathfrak{c}) = W$. If W is a good word, and if $i \in \mathbb{N}$ is the length of the shortest path (excluding time elapse transitions) from W to $W_\emptyset$, then we say that dist(W) is i. Given a set R of words, dist(R) $\in \mathbb{N} \cup \{\infty\}$ is defined as the length of the shortest path (excluding time elapse transitions) from some $W \in R$ to $W_\emptyset$. More precisely, if $R = \emptyset$, then dist(R) $= \infty$, otherwise, dist(R) $= min_{W \in R}$dist(W).

▶ **Lemma 9.** **1.** $Good($Processed $\cup$ Next$) \rightarrow Good(\mathcal{W}_0)$
2. $Good(\mathcal{W}_0) \rightarrow$ dist(Processed) $>$ dist(Next)

To prove the invariants, we use the following lemma.

▶ **Lemma 10.** *If $W \preceq W'$ and* dist(W') $= i$, *then* dist(W) $= j$ *for some* $j \leq i$.

Due to the well-quasi ordering, the algorithm terminates: if not, over a period of time, there will be an infinite sequence of words in Next, each new word added having the property that it does not dominate any of its predecessors. This would constitute an infinite non saturating sequence, directly contradicting Higman's Lemma. The algorithm returns FALSE only when Next is empty. Then, dist(Processed) $>$ dist(Next) is not true. Therefore, by invariant 2 in lemma 9, $\mathcal{W}_0$ is not good. The algorithm returns TRUE only if either $W_\emptyset$ is already in $\mathcal{W}_0$, or if $W_\emptyset$ is a member of Minimize(succ(W)) for some $W \in$ Next. In either case, Next is good. Then, by invariant 1 of lemma 9, $\mathcal{W}_0$ is good. This gives the following lemma.

▶ **Lemma 11.** *Algorithm* REACH EMPTY *terminates and returns true iff starting from the initial configuration $\mathfrak{c}_0$ in $\mathcal{M}$, $\mathfrak{c}_{goal}$ is reachable.*

This concludes the proof of theorem 5. A detailed discussion with proofs for the lemmas can be found in the extended version of the paper [1].

Notice that the stateless, and time-independent properties of $\mathcal{M}$ are crucial in Lemma 13. The example below shows that relaxing either condition violates lemma 13.

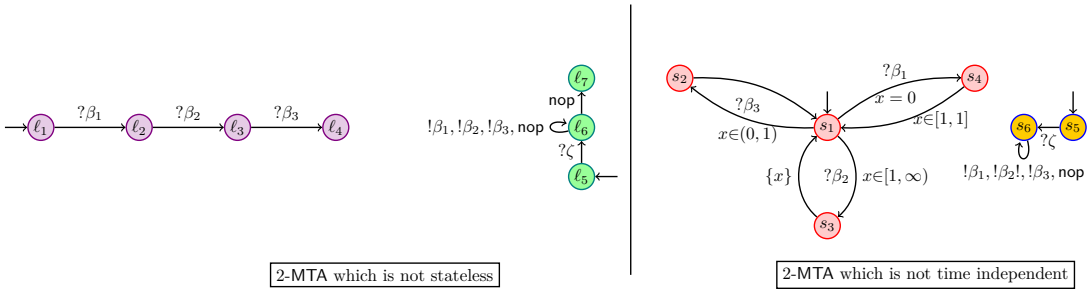

Figure 3

To the left is a 2-MTA which is not stateless. It can be seen that $\mathfrak{c}_1{=}(\ell_1, \ell_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (\ell_1, \ell_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset){=}\mathfrak{c}_2$. Hence, $\mathsf{W}(\mathfrak{c}_1) \preceq \mathsf{W}(\mathfrak{c}_2)$. Indeed from $\mathfrak{c}_2$, one can reach $(\ell_4, \ell_6, \emptyset, \emptyset)$, but not from $\mathfrak{c}_1$. To the right is a 2-MTA which is not time independent. It can be seen that $\mathfrak{c}_1{=}(((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset) \preceq (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset){=}\mathfrak{c}_2$. However, $(((s_1, 0), s_6), \emptyset, \emptyset)$ is reachable from $\mathfrak{c}_2$ but not from $\mathfrak{c}_1$.

6 Conclusion

We proposed a model to address the verification problem for timed asynchronous programs. We identified a special subclass (stateless and time-independent) for which the reachability problem is decidable and control reachability is PSPACE-complete. There are multiple avenues for further work. The first question is to check the tightness of the EXPSPACE lower bound provided. Another question would be to consider the model where we use *priority bags* instead of bags. In a priority bag, tasks have associated deadlines and priorities. The process, while picking up a task for execution, is expected to pick up a task with the highest priority. Queues are yet another interesting data structure in place of bags: in this set up, the tasks which require a processor's attention are picked up in the order in which they were assigned by various processes. One can also look at mutiset timed pushdown systems, which extend the model of [16] with time, and multiple processes. Finally, we can move from the one player setting to two players, where the environment chooses a task for the process to execute. Under this two player setting, the question would be if the system has a strategy to execute all the pending tasks.

References

1 P.A. Abdulla, M. Faouzi Atig, S. Krishna, and S. Vaidya. *Verification of Timed Asynchronous Programs*. `http://www.cse.iitb.ac.in/~krishnas/fsttcs2018.pdf`.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, April 1994.

3 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Verification of asynchronous programs with nested locks. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 11:1–11:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

4 Mohamed Faouzi Atig, Ahmed Bouajjani, and Tayssir Touili. Analyzing asynchronous programs with preemption. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India*, volume 2 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008.

5 Rohit Chadha and Mahesh Viswanathan. Decidability results for well-structured transition systems with auxiliary storage. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings*, volume 4703 of *Lecture Notes in Computer Science*, pages 136–150. Springer, 2007.

6 Michael Emmi, Shaz Qadeer, and Zvonimir Rakamaric. Delay-bounded scheduling. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 411–422. ACM, 2011.

7 Pierre Ganty and Rupak Majumdar. Analyzing real-time event-driven programs. In Joël Ouaknine and Frits W. Vaandrager, editors, *Formal Modeling and Analysis of Timed Systems, 7th International Conference, FORMATS 2009, Budapest, Hungary, September 14-16, 2009. Proceedings*, volume 5813 of *Lecture Notes in Computer Science*, pages 164–178. Springer, 2009.

8 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012.

9 Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The ordinal-recursive complexity of timed-arc petri nets, data nets, and other enriched nets. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 355–364. IEEE Computer Society, 2012.

10 Graham Higman. Ordering by divisibility in abstract algebras. *Proceedings of the London Mathematical Society*, 3(1):326–336, 1952.

11 Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 339–350. ACM, 2007.

12 Joseph B Kruskal. The theory of well-quasi-ordering: A frequently discovered concept. *Journal of Combinatorial Theory, Series A*, 13(3):297–305, 1972.

13 Pallavi Maiya, Rahul Gupta, Aditya Kanade, and Rupak Majumdar. Partial order reduction for event-driven multi-threaded programs. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 680–697. Springer, 2016.

14 Rupak Majumdar and Zilong Wang. Bbs: A phase-bounded model checker for asynchronous programs. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 496–503. Springer, 2015.

15 M. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

16 Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In Thomas Ball and Robert B. Jones, editors, *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 300–314. Springer, 2006.

17 Jiří Srba. Timed-arc petri nets vs. networks of timed automata. *In Proceedings of the 26th International Conference on Application and Theory of Petri Nets (ICATPN 2005). Netherlands: Springer-Verlag, 2005*, pages 385–402, 2005.

Appendix

A Missing Proofs from Section 4

In this section, we give the missing proofs from section 4.

A.1 Proof of Theorem 1

The formal construction is as follows: Given the N-MTA $\mathcal{M}$, we construct the TPN with read arcs $\mathcal{N} = (P, T, F, c)$ with an initial marking M_0, whose coverability is equivalent to the control state reachability of $\mathcal{M}$. The initial marking is one which adds a token to places $p_{\ell_1^0}, \ldots, p_{\ell_N^0}$, one each from L_i^0, and tokens are added into places simulating multisets according to the initial configuration of the N-MTA.

- The set of places

$$P = \{p_\ell \mid \ell \in \cup_{i=1}^{N} L_i\} \cup \{p_x \mid x \in \mathcal{X}\} \cup \{p^i_{(a,d)} \mid 1 \leq i \leq N, a \in \Sigma, d \in \{0, 1, \ldots, d_{max}, \infty\}\}$$

- The set of transitions

$$T = (\{(\ell, g, i?a, Y, \ell') \in E_i, 1 \leq i \leq N\} \times \{0, 1, \ldots, d_{max}, \infty\}) \cup \{(\ell, g, i!j(a[d]), Y, \ell'), (\ell, g, \mathsf{nop}_i, Y, \ell') \in E_i \mid 1 \leq i \leq N\}$$

- Consider the transition $t = (\ell, g, i!j(a[d]), Y, \ell') \in E_i$ in automaton T_i. Let $g(x)$ be the constraints pertaining to clock x. The constraints on arcs are given using the function c.
  - We add arcs $(p_\ell, t), (t, p_{\ell'})$ for the change in control location, and $(t, p^j_{(a,d)})$ for the task and $c(p_\ell, t) = [0, \infty)$.
  - For each clock $x \in \mathcal{X}_i$, if $x \in Y$, then we add the arcs (p_x, t) and (t, p_x) and $c(p_x, t) = g(x)$. These arcs essentially simulate resetting the clock x.
  - For each clock $x \in \mathcal{X}_i$, if $x \notin Y$, then add the read arc $(p_x, t)*$ and $c(p_x, t)* = g(x)$. The read arc simulates checking the clock constraint for clock x.
- Consider the transition $t = (\ell, g, i?a, Y, \ell') \in E_i$ in automaton T_i.
  - We add the arcs $(p_\ell, (t, d))$, $((t, d), p_{\ell'})$, $(p^i_{(a,d)}, (t, d))$ for every $d \in \{0, 1, \ldots, d_{max}, \infty\}$. Also, $c(p_\ell, (t, d)) = [0, \infty)$, $c(p^i_{(a,d)}, (t, d)) = [0, d]$ for each $d \in \{1, \ldots, d_{max}\}$. Note that, we add arcs for each deadline value, since while picking a task from the multiset, it could be any possible deadline value.
  - The clock constraints and resets are handled by adding arcs (and read arcs) between places p_x and (t, d) for each $d \in \{0, 1, \ldots, d_{max}, \infty\}$ as seen above.
- Consider the transition $t = (\ell, g, \mathsf{nop}_i, Y, \ell') \in E_i$ in automaton T_i.
  - We add the arcs (p_ℓ, nop_i), $(\mathsf{nop}_i, p_{\ell'})$. Also, $c(p_\ell, \mathsf{nop}_i) = [0, \infty)$.
  - The clock constraints and resets are handled by adding arcs (and read arcs) between places p_x and nop_i as seen above.

  Time elapse transitions are simulated according to the semantics of timed petri nets.

Note that the configuration of the N-MTA at any point is simulated correctly by the corresponding marking of the timed petri net. Also, by construction, corresponding to each T_i, there is one and only one place p_ℓ in the net, such that $\ell \in L_i$, is marked at any point in any execution. The control reachability of $\overline{f} = (f^1, \ldots, f^N)$ in the N-MTA $\mathcal{M}$ thus, reduces

to the coverability of the marking $\{p_{f^1}, \ldots, p_{f^N}\}$ in the constructed timed petri net with read arcs and therefore is decidable. Also, since the reduction is polynomial in the size of the N-MTA, the complexity of the problem is the same as that of coverability in timed petri nets. [9] has shown that the complexity of the coverability problem for timed petri nets at the level $F_{\omega^{\omega^\omega}}$ of the fast growing hierarchy. Our reduction shows that the control state reachability in MTA is bounded above by the same complexity.

A.2 Formal Construction of the 1-safe TPN in Theorem 3

The formal construction is as follows: Given the N-MTA $\mathcal{M}$, we construct a 1-safe TPN with read arcs $\mathcal{N} = (P, T, F, c)$ with an initial marking M_0, whose coverabiltiy is equivalent to the control state reachability of $\mathcal{M}$.

- The set of places

$$P = \{p_\ell \mid \ell \in \cup_{i=1}^{N} L_i\} \cup \{p_x \mid x \in \mathcal{X}\} \cup \{p^i_{(a,d,z)} \mid 1 \leq i, z \leq N, a \in \Sigma, d \in \{0, 1, \ldots, d_{max}, \infty\}\}$$

- The set of transitions

$$\begin{aligned} T = &(\{(\ell, g, i?a, Y, \ell') \in E_i, 1 \leq i \leq N\} \times \{1, \ldots, N\} \times \{0, 1, \ldots, d_{max}, \infty\}) \cup \\ &(\{(\ell, g, i!j(a[d]), Y, \ell') \in E_i \mid 1 \leq i \leq N\} \times \{1, \ldots, N+1\}) \cup \\ &\{(\ell, g, \mathsf{nop}_i, Y, \ell') \in E_i \mid 1 \leq i \leq N\} \end{aligned}$$

The initial marking adds a token in places $p_{\ell_1^0}, \ldots, p_{\ell_N^0}$, (one ℓ_0^i from each L_0^i) and *may* add a token in places corresponding to tasks in the initial marking. This choice is made non-deterministically based on which of the initial tasks are relevant.

The main difference in this construction, resulting in a 1-safe TPN with respect to the construction in Theorem 1 are the places $p^i_{(a,d,z)}$ rather than places $p^i_{(a,d)}$ as in Theorem 1. The bound proved on the relevant tasks ensures that there is at most one token in places $p^i_{(a,d,z)}$ in this construction, as opposed to the absence of any bound on the number of tokens in places $p^i_{(a,d)}$ in Theorem 1.

- Consider the transition $t = (\ell, g, i!j(a[d]), Y, \ell') \in E_i$ in automaton T_i. Let $g(x)$ be the constraints pertaining to clock x. The constraints on arcs are given using the function c.
  - We add arcs $(p_\ell, (t, z))$, $((t, z), p_{\ell'})$, $((t, z), p^j_{(a,d,z)})$, for $1 \leq z \leq N$ for the case when the task is relevant, and the arcs $(p_\ell, (t, N+1))$, $((t, N+1), p_{\ell'})$, for the case when the task is not relevant. Note that one of the transitions (t, z), for $1 \leq z \leq N+1$ is non-deterministically chosen to simulate the transition $(\ell, g, i!j(a[d]), Y, \ell')$.
  - If $x \in Y$, we add the arcs $(p_x, (t, z))$ and $((t, z), p_x)$ for $1 \leq z \leq N+1$ and $c(p_x, (t, z)) = g(x)$.
  - If $x \notin Y$, then add the read arcs $(p_x, (t, z))*$ for $1 \leq z \leq N+1$ and $c(p_x, (t, z))* = g(x)$ (* denotes a read arc).
  - Finally, for $1 \leq z \leq N+1$, $c(p_\ell, (t, z)) = [0, \infty)$.
- Consider the transition $t = (\ell, g, i?a, Y, \ell') \in E_i$ in automaton T_i.
  - We add arcs $(p_\ell, (t, z, d))$, $((t, z, d), p_{\ell'})$, $(p^j_{(a,d,z)}, (t, z, d))$, and $c(p^j_{(a,d,z)}, (t, z, d)) = [0, d]$ for $1 \leq z \leq N$, $d \in \{0, 1, \ldots, d_{max}, \infty\}$. Note that one of the transitions (t, z, d), for $1 \leq z \leq N$, $d \in \{0, 1, \ldots, d_{max}, \infty\}$, if enabled, is non-deterministically chosen to simulate the transition $(\ell, g, i?a, Y, \ell')$.

- If $x \in Y$, we add the arcs $(p_x, (t, z, d))$ and $((t, z, d), p_x)$ for $1 \leq z \leq N$, $d \in \{0, 1, \ldots, d_{max}, \infty\}$ and $c(p_x, (t, z, d)) = g(x)$.
- If $x \notin Y$, then add the read arcs $(p_x, (t, z, d))*$ for $1 \leq z \leq N$, $d \in \{0, 1, \ldots, d_{max}, \infty\}$ and $c(p_x, (t, z, d))* = g(x)$.
- Finally, for $1 \leq z \leq N$, $d \in \{0, 1, \ldots, d_{max}, \infty\}$, $c(p_\ell, (t, z, d)) = [0, \infty)$.

- Consider the transition $t = (\ell, g, \mathsf{nop_i}, Y, \ell') \in E_i$ in automaton T_i.
  - We add the arcs $(p_\ell, \mathsf{nop_i})$, $(\mathsf{nop_i}, p_{\ell'})$. Also, $c(p_\ell, \mathsf{nop_i}) = [0, \infty)$.
  - The clock constraints and resets are handled by adding arcs (and read arcs) between places p_x and $\mathsf{nop_i}$ as seen above.

  Time-elapse transitions are simulated according to the semantics of timed petri nets.

B Missing Proofs from Section 5

In this section, we give the missing proofs of the lemmas from section 5.

B.1 Proof of Theorem 4

Proof. The proof is done by reduction from the reachability problem for an 2-counter machines (which is known to be undecidable). The main idea is to construct an 1-MTA whose set of states contains the states of the two counter machine plus some auxiliary states that are used to simulate the zero tests as we will see later on. The 1-MTA has two types of tasks a and b. The number of pending tasks of type a (resp. b) corresponds to the value of the counter c_1 (resp. c_2). Furthermore, the 1-MTA has one clock x that is used to check that no time elapsed when simulating some transitions of the two counter machine.

To simulate an increment transition of the form $(q, c_1{+}{+}, q')$ (resp.$(q, c_2{+}{+}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally adds a pending task of type a (resp. b) with zero as its deadline. Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a decrement transition of the form $(q, c_1{-}{-}, q')$ (resp. $(q, c_2{-}{-}, q')$) of the two counter machine, the 1-MTA proceeds as follows: first it checks that the value of the clock $x = 0$, then it will change its state from q to q' and finally consumes a pending task of type a (resp. b). Observe that we need only one transition to perform all these steps of the simulation of an increment operation.

To simulate a zero test transition of the form $(q, c_1 == 0, q')$ (resp. $(q, c_2 == 0, q')$) of the two counter machine, the 1-MTA proceeds as follows: (i) it checks that the value of the clock $x = 0$, (ii) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to one time unit, (iii) it will change its state from q to q', (iv) it checks that the value of the clock x is still zero, (v) it checks that one time unit has elapsed (ie., checking whether $x \in [1, 1]$) and resets the clock x, (vi) it enters to a loop where it consumes a task of type b (resp. a) and creates a task of the same type but its deadline is now set to zero, and (vii) it checks that value of $x = 0$. Here the auxiliary states are needed in the simulation of these steps.

Observe that if the 1-MTA reaches the final state with empty set of pending tasks, then all the simulation of the zero tests are performed correctly. Finally, note that the constructed 1-MTA is time independent. ◀

B.2 Proof of Lemma 7

Let $\mathfrak{c}_1, \mathfrak{c}_2$ be configurations such that $\mathfrak{c}_1 \sim \mathfrak{c}_2$. Let $\mathfrak{c}_1'$ be a configuration such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}_1'$ for some transition $e \in E_i$. We have to show that there exists a configuration $\mathfrak{c}_2'$ such that $\mathfrak{c}_2 \xrightarrow{e} \mathfrak{c}_2'$ and $\mathfrak{c}_1' \sim \mathfrak{c}_2'$.

Let $\mathfrak{c}_1{=}(\overline{q}, \overline{m})$ and $\mathfrak{c}_2{=}(\overline{q'}, \overline{m'})$. Let $\overline{q}{=}((\ell_1, \nu_1), \ldots, (\ell_N, \nu_N))$, $\overline{q'}{=}((\ell_1', \nu_1'), \ldots, (\ell_N', \nu_N'))$, $\overline{m}{=}(M_1, \ldots, M_N)$ and $\overline{m'}{=}(M_1', \ldots, M_N')$. Since $\mathfrak{c}_1 \sim \mathfrak{c}_2$, we have

1. $\ell_i' = \ell_i$ for all $1 \leq i \leq N$,
2. the clock valuations ν_i and ν_i' are equivalent for all $1 \leq i \leq N$ (that is, belong to the same region),
3. For each task tuple $(a, r, d) \in M_i$, there exists a task tuple $(a, r', d) \in M_i'$ such that $r, r' \in \mathbb{R}$ are in the same region,
4. For each task tuple $(a, r', d) \in M_i'$, there exists a task tuple $(a, r, d) \in M_i$ such that $r, r' \in \mathbb{R}$ are in the same region.

The clock constraint along the transition e is satisfied in $\mathfrak{c}_1$ iff it is satisfied in $\mathfrak{c}_2$ since the integer parts of all clocks agree in all ν_i, ν_i' for all $1 \leq i \leq N$, and the ordering of the fractional parts of all clocks in ν_i, ν_i' are the same. The clocks which are reset (if any) also agree in both ν_i, ν_i' after the transition, preserving the equivalence of the valuations. Likewise, a task can be executed on the transition from $\mathfrak{c}_1$ iff it can be executed on the transition from $\mathfrak{c}_2$, since the names of the executed tasks are the same, and their ages lie in the same region.

Next, we show that if $\mathfrak{c}_1 \xrightarrow{t} \mathfrak{c}_1'$, then there exists a time $t' \in \mathbb{R}$ such that $\mathfrak{c}_2 \xrightarrow{t'} \mathfrak{c}_2'$ and $\mathfrak{c}_1' \sim \mathfrak{c}_2'$. Since ν_i, ν_i' belong to the same region, indeed given t, one can find t' such that $\nu_i + t$ and $\nu_i' + t'$ are also equivalent. The same argument applies to the ages r, r' of tasks in all multisets. Thus, indeed $\mathfrak{c}_2 \xrightarrow{t'} \mathfrak{c}_2'$ with $\mathfrak{c}_1' \sim \mathfrak{c}_2'$.

The converse direction follows exactly in a similar way.

B.3 Proof of Lemma 8

We first state the following lemma, whose proof is easy to see from Lemma 7.

▶ **Lemma 12.** *For $W_1, W_2 \in \mathcal{W}$, and $e \in E_i$, $W_1 \xrightarrow{e} W_2$ iff there exist configurations $\mathfrak{c}_1 \in \mathsf{W}^{-1}(W_1)$ and $\mathfrak{c}_2 \in \mathsf{W}^{-1}(W_2)$ such that $\mathfrak{c}_1 \xrightarrow{e} \mathfrak{c}_2$. The case of a time elapse transition is similar.*

Given a word W, it is easy to construct a configuration $\mathfrak{c}$ such that $\mathsf{W}(\mathfrak{c}) = W$. Then given any transition $e \in E_i$, there are finitely many configurations $\mathfrak{c}'$ such that $\mathfrak{c} \xrightarrow{e} \mathfrak{c}'$ is enabled. This list is easily computed.

Next, we look at time elapse transitions. For $t \in \mathbb{R}$, $\mathsf{W}(\mathfrak{c} + t)$ is a word which has the same number of letters as W, the finite collection of which is easy to compute.

For each word W, and each transition e, computing the successor with respect to e is done by examining some configuration $\mathfrak{c} \in \mathsf{W}^{-1}(W)$. By Lemma 12, the choice of a particular configuration is not important while computing successors. Since the function W which converts configurations $\mathfrak{c}$ to words in $\mathcal{W}$ is computable, the above is an effective procedure to compute $\mathsf{succ}(W)$.

B.4 Proving the Invariants of Algorithm 1

Before we prove the invariants, we prove the following lemma.

▶ **Lemma 13.** *If $W \preceq W'$ and $\mathsf{dist}(W') = i$, then $\mathsf{dist}(W) = j$ for some $j \leq i$.*

Proof. In the following, we refer to W' as i-good whenever $\mathsf{dist}(W') = i$, and W as j-good whenever $\mathsf{dist}(W) = j$.

Assume $W \preceq W'$ and W' is i-good for some $i \geq 1$. Then we know that the length of the shortest path from W' to $W_\emptyset$ is i. Thus, i discrete transitions suffice to reach $W_\emptyset$ from W'. Thanks to the bisimulation lemma 7 and lemma 12, corresponding to each run from W' to $W_\emptyset$ in $\mathcal{W}$, we can find a run in $\mathcal{M}$ from some configuration in $\mathsf{W}^{-1}(W')$ to a configuration in $\mathsf{W}^{-1}(W_\emptyset)$. The rest of the proof considers ρ as a run in $\mathcal{M}$ from $\mathsf{W}^{-1}(W')$ to $\mathsf{W}^{-1}(W_\emptyset)$. Let $\mathfrak{c}' = (\overline{q'}, \overline{m'}) \in \mathsf{W}^{-1}(W')$ and $\mathfrak{c} = (\overline{q}, \overline{m}) \in \mathsf{W}^{-1}(W')$. Then

1. $\overline{q} = ((\ell_1, \nu_1), \ldots, (\ell_N, \nu_N))$, $\overline{q'} = ((\ell_1, \nu'_1), \ldots, (\ell_N, \nu'_N))$,
2. ν_i and ν'_i are equivalent for all $1 \leq i \leq N$,
3. $\overline{m'} = (M'_1, \ldots, M'_N)$, $\overline{m} = (M_1, \ldots, M_N)$, $M_i \subseteq M'_i$ for all $1 \leq i \leq N$.

Let ρ be the run from $\mathfrak{c}'$ to $\mathfrak{c}_{goal} \in \mathsf{W}^{-1}(W_\emptyset)$. Let ρ_i be the projection of ρ, where all transitions are only pertaining to T_i. Since all tasks in the multiset M'_i are completely executed before reaching $\mathfrak{c}_{goal}$, there are at least $|M'_i|$ blocks in ρ_i, each block picking up a task.

Since $\mathcal{M}$ is stateless and time independent, at the beginning of each block which executes a task tuple from M'_i, the location of T_i is a unique $\ell_i^0 \in L_0^i$. Since each block of ρ_i begins with a transition of the form $i?a$, does not check any constraints on this transition, and resets all clocks, a subsequent block is affected only by the time elapse so far, and the clock valuations in earlier blocks have no role to play. Since $\mathfrak{c}'$ indeed reaches $\mathfrak{c}_{goal}$, all the tasks in M'_i are executed before they grow too old, satisfying their deadline. Let $\alpha_1, \ldots, \alpha_h$ be the blocks along ρ_i. Choose indices $i_1, \ldots, i_z$ from $1, 2, \ldots, h$ such that $\alpha_{i_1}, \ldots, \alpha_{i_z}$ are the blocks where the tasks of $M_i \subseteq M'_i$ are executed.

Since ν_i and ν'_i are equivalent clock valuations, the transitions that are enabled along ρ_i can also be enabled in a run which starts from a state (ℓ_i, ν_i). Thus, starting from state (ℓ_i, ν_i), we can proceed exactly as in run ρ_i. The blocks of ρ_i outside of $\alpha_{i_1}, \ldots, \alpha_{i_z}$ can be omitted, since these omitted blocks execute tasks in $M'_i - M_i$. Let $i_1 < \cdots < i_z$. If a block outside of $\alpha_{i_1}, \ldots, \alpha_{i_z}$ adds some task a to the bag of T_i, and if it is processed in a block β_a, then we also omit the block β_a; likewise, if block β_a adds a task b to the bag of T_i, then we omit the block γ_b where b is executed and so on. This way, we obtain a run ρ'_i which executes all tasks in M_i, and has fewer blocks than ρ_i. Indeed, the length (the number of discrete transitions) in ρ'_i is smaller than that in ρ_i.

Now, we show that we can construct a run ρ' instead of ρ which also reaches a configuration where the bag of T_i is empty, such that the number of discrete transitions in ρ' is smaller than in ρ. The idea is similar to what we did to obtain ρ'_i from ρ_i : the extra thing is that when we omit a block from ρ_i, if that block adds a task c to the bag of T_j, $j \neq i$, then we will also omit the block where T_j executes c; further, all tasks added in this block will be executed in subsequent blocks, which should be removed and so on. In summary, to obtain ρ' from ρ, we do the following.

1. Break ρ into blocks. Let $\alpha_1, \ldots, \alpha_m$ be the block labels in ρ. Each block α_i begins with executing some task. For each block α_i, let $\mathsf{child}(\alpha_i)$ be the set of blocks which execute the set of tasks which have been created in block α_i.
2. For each multiset M_i, $1 \leq i \leq N$, let $p_i = |M_i|$. Let $\alpha_{z_{i_1}}, \ldots, \alpha_{z_{p_i}}$ be the blocks of ρ which execute a task in M_i.
3. Omit all blocks other than $\alpha_{z_{i_1}}, \ldots, \alpha_{z_{p_i}}$ for all $1 \leq i \leq N$.

4. If α is an omitted block, then omit blocks $\mathsf{child}(\alpha)$ also from ρ. Notice that the task executed in any of the blocks $\mathsf{child}(\alpha)$ is not from M_i.
5. The resultant run ρ has transitions labeled $\alpha_{z_{i_1}}, \ldots, \alpha_{z_{p_i}}$, and $\mathsf{child}(\alpha_{z_{i_1}}), \ldots, \mathsf{child}(\alpha_{z_{p_i}})$ for all $1 \leq i \leq N$. All the time elapse transitions in ρ appear in the new run ρ'.
6. The execution of each block in ρ' proceeds exactly as it did in ρ : this is so, since there are no dependencies of clock valuations across blocks; only the time elapse that has happened before a block commences matters. Since
   - the time elapses are preserved, and
   - clocks constraints are not checked at the start of a block, and
   - since all clocks are reset to zero after the start of a block,

   omitting a block has no side effects, and all the relevant blocks in ρ' proceed successfully, and we reach the configuration where all the multisets are empty, just as we did in ρ.
7. Clearly, the new run ρ' has fewer discrete transitions than ρ. Since the run ρ' begins with $\mathfrak{c} \in \mathsf{W}^{-1}(W)$, and ρ with $\mathfrak{c}' \in \mathsf{W}^{-1}(W')$, if W' is i-good for some $i \in \mathbb{N}$, then W' is j-good for some $j \leq i$.

◀

Stateless-ness, Time-independence of $\mathcal{M}$ in lemma 13

Notice that the stateless, and time-independent properties of $\mathcal{M}$ are crucial in Lemma 13. The example below shows that relaxing either condition violates lemma 13.

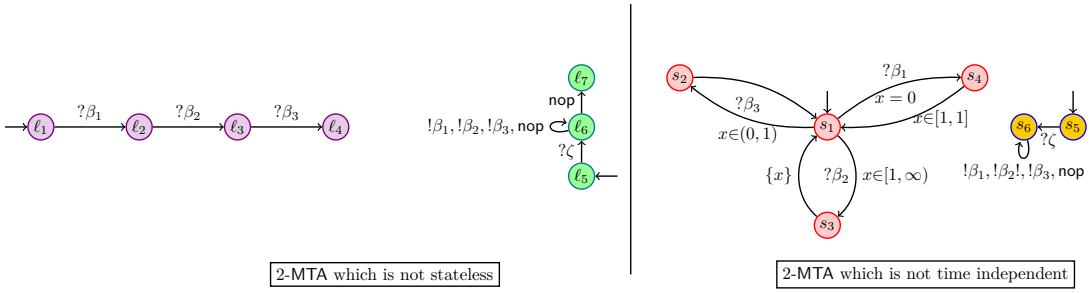

Figure 4

To the left is a 2-MTA which is not stateless. It can be seen that $\mathfrak{c}_1 = (\ell_1, \ell_6, \{\beta_1, \beta_3\}, \emptyset) \preceq (\ell_1, \ell_6, \{\beta_1, \beta_2, \beta_3\}, \emptyset) = \mathfrak{c}_2$. Hence, $\mathsf{W}(\mathfrak{c}_1) \preceq \mathsf{W}(\mathfrak{c}_2)$. Indeed from $\mathfrak{c}_2$, one can reach $(\ell_4, \ell_6, \emptyset, \emptyset)$, but not from $\mathfrak{c}_1$. To the right is a 2-MTA which is not time independent. It can be seen that $\mathfrak{c}_1 = (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset) \preceq (((s_1, 0), s_6), \{(\beta_1, 0, \infty), (\beta_2, 0, \infty), (\beta_3, 0, \infty)\}, \emptyset) = \mathfrak{c}_2$. However, $(((s_1, 0), s_6), \emptyset, \emptyset)$ is reachable from $\mathfrak{c}_2$ but not from $\mathfrak{c}_1$.

B.5 Proof of Lemma 9

Now, we prove the invariants. The invariants are as follows.

1. $Good(\mathsf{Processed} \cup \mathsf{Next}) \rightarrow Good(\mathcal{W}_0)$
2. $Good(\mathcal{W}_0) \rightarrow \mathsf{dist}(\mathsf{Processed}) > \mathsf{dist}(\mathsf{Next})$

Proof. It is trivial to see that the invariants hold good at the beginning of the loop. We show that the invariants continue to hold good when the loop body is executed from a configuration of the algorithm in which the invariants hold. We use $\mathsf{Processed}^{old}$ and Next^{old} to denote the values of Processed and Next before executing the loop body, and we use $\mathsf{Processed}^{new}$ and Next^{new} to denote their values when the control gets back to the head of the loop after executing the loop body once. We assume $\mathsf{Next}^{old} \neq \emptyset$.
Let us start with invariant 1. Assume $\mathit{Good}(\mathsf{Processed}^{old} \cup \mathsf{Next}^{old})$ holds. The very first time, we know $\mathsf{Processed}^{old} = \emptyset$ and Next^{old} is a minimized version of $\mathcal{W}_0$. Clearly, $\mathcal{W}_0$ is good. In subsequent steps, the minimized versions of successors added to Next ensure that $\mathit{Good}(\mathsf{Processed}^{new} \cup \mathsf{Next}^{new})$ is true. Hence, invariant 1 is true.

If we assume $\neg\mathit{Good}(\mathsf{Processed}^{old} \cup \mathsf{Next}^{old})$, then clearly, $W_\emptyset$ is not reachable, and the same holds good after the successor computation and minimization when things are added to Next. Then we have $\neg\mathit{Good}(\mathsf{Processed}^{new} \cup \mathsf{Next}^{new})$, and invariant 1 holds.

We proceed to invariant 2. Assume that $\mathit{Good}(\mathcal{W}_0)$ holds (the other case is trivial). Then $\mathsf{dist}(\mathsf{Processed}^{old}) > \mathsf{dist}(\mathsf{Next}^{old})$. We distinguish two cases.

1. The word W removed from Next^{old} is such that $\mathsf{dist}(W) = \infty$, or there is a word V in $\mathsf{Processed}^{old}$ such that $\mathsf{dist}(V) \leq \mathsf{dist}(W)$. In this case, dist(Processed) will not decrease in step 1 inside the loop. From $\mathsf{dist}(\mathsf{Processed}^{old}) > \mathsf{dist}(\mathsf{Next}^{old})$, there is some word $W' \in \mathsf{Next}^{old}$ such that $\mathsf{dist}(W') = \mathsf{dist}(\mathsf{Next}^{old}) < \mathsf{dist}(\mathsf{Processed}^{old}) \leq \mathsf{dist}(V) \leq \mathsf{dist}(W)$. Then, dist(Next) also does not change in step 1 inside the loop. Next, for any word U, removing V such that $U \preceq V$ from Next and Processed in step (c) and adding U to Next in step (d) cannot falsify $\mathsf{dist}(\mathsf{Processed}^{new}) > \mathsf{dist}(\mathsf{Next}^{new})$, since $\mathsf{dist}(U) \leq \mathsf{dist}(V)$ by lemma 13. Hence, invariant 2 must hold good for $\mathsf{Processed}^{new}$ and Next^{new} too.
2. The word W removed from Next^{old} is such that $\mathsf{dist}(W) \neq \infty$, and there is no word V in $\mathsf{Processed}^{old}$ such that $\mathsf{dist}(V) \leq \mathsf{dist}(W)$. In this case, the value of dist(Processed) decreases to $\mathsf{dist}(W)$ in step 1 inside the loop. Clearly, $\mathsf{dist}(W) \neq 0$, or we would have terminated before. Then there must be a successor W' of W which is either $W_\emptyset$, and the loop stops, or is one step closer to reaching $W_\emptyset$, giving $\mathsf{dist}(W') < \mathsf{dist}(W)$. W' is either a part of Next^{new}, or there is a word $W'' \in \mathsf{Next}^{old}$ such that $W'' \preceq W'$, implying $\mathsf{dist}(\mathsf{Processed}^{new}) > \mathsf{dist}(\mathsf{Next}^{new})$. It is not possible that there is a word $W'' \in \mathsf{Processed}^{old}$ such that $W'' \preceq W'$, since we know that for all words W'' in $\mathsf{Processed}^{old}$, $\mathsf{dist}(W'') > \mathsf{dist}(W) > \mathsf{dist}(W')$ (from lemma 13, we know that $W'' \preceq W'$ implies $\mathsf{dist}(W'') \leq \mathsf{dist}(W')$). If some word is removed from Processed in step (c), then dist(Processed) can only increase, and hence we are done.

◀