Model checking time-constrained scenario-based specifications*

S. Akshay[1,2], Paul Gastin[1], Madhavan Mukund[2], and K. Narayan Kumar[2]

1 **LSV, ENS Cachan, INRIA, CNRS, France**
{akshay,Paul.Gastin}@lsv.ens-cachan.fr
2 **Chennai Mathematical Institute, Chennai, India**
{madhavan,kumar}@cmi.ac.in

Abstract

We consider the problem of model checking message-passing systems with real-time requirements. As behavioural specifications, we use message sequence charts (MSCs) annotated with timing constraints. Our system model is a network of communicating finite state machines with local clocks, whose global behaviour can be regarded as a timed automaton. Our goal is to verify that all timed behaviours exhibited by the system conform to the timing constraints imposed by the specification. In general, this corresponds to checking inclusion for timed languages, which is an undecidable problem even for timed regular languages. However, we show that we can translate regular collections of time-constrained MSCs into a special class of event-clock automata that can be determinized and complemented, thus permitting an algorithmic solution to the model checking problem.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.204

1 Introduction

In a distributed system, several agents interact to generate a global behaviour. This interaction is usually specified in terms of scenarios, using message sequence charts (MSCs) [8]. Protocol specifications typically include timing requirements for messages and descriptions of how to recover from timeouts, so a natural and useful extension to MSCs is to add timing constraints between pairs of events, yielding time-constrained MSCs (TCMSCs).

Infinite collections of MSCs are typically described using message sequence graphs (MSGs). An MSG, a finite directed graph with nodes labelled by MSCs, is the most basic form of a High-level Message Sequence Chart (HMSC) [9]. We generalise MSGs to time-constrained MSGs (TCMSGs), where nodes are labelled by TCMSCs and edges may have additional time constraints between nodes.

A natural system model in this setting is a timed message-passing automaton (timed MPA), a set of communicating finite-state machines equipped with clocks that are used to guard transitions, as in timed automata [4]. Just as the runs of timed automata are described in terms of timed words, the interactions exhibited by timed MPAs can be described using timed MSCs—MSCs in which each event is assigned an explicit timestamp. The global state space of a timed MPA defines a timed automaton and in this paper we focus on this simplified global view of timed message-passing systems, though our results go through smoothly for the distributed system model as well.

* Supported by ANR-06-SETI-003 DOTS, ARCUS Île de France-Inde, CMI-TCS Academic Alliance.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 204–215

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our aim is to check if all timed MSCs accepted by a timed MPA conform to the time constraints given by a TCMSG specification. To make the problem tractable, we focus on *locally synchronized* TCMSGs—those for which the underlying behaviour is guaranteed to be regular [7]. In general, our model checking problem corresponds to checking inclusion for timed languages, which is known to be undecidable even for timed regular languages [2]. Fortunately, it turns out that timing constraints in a TCMSG correspond to a very restricted use of clocks. This allows us to associate with each TCMSG an (extended) event clock automaton that accepts all timed MSCs that are consistent with the timing constraints of the TCMSG. These event clock automata can be determinized and complemented, yielding an algorithmic solution to our model checking problem.

The paper is organized as follows. We begin with some preliminaries where we introduce (timed) MSCs and MSGs and state the model-checking problem. In Section 3 we introduce MSC event clock automata and show that they can be determinized and complemented. The next section has the main technical result: translating locally synchronized TCMSGs to finite state MSC event clock automata, which yields a solution to the model-checking problem in Section 5.

2 Preliminaries

2.1 Message sequence charts

A *message sequence chart (MSC)* describes the messages exchanged between a set *Proc* of processes in a distributed system. The first diagram in Figure 1 is an MSC involving two users and a server. Each process evolves vertically along a lifeline. Messages are shown by arrows between the lifelines of the sender and receiver.

Each message consists of two *events*, send and receive, and is labelled using a finite set of message labels. For instance, the events u_1 and a_1 are the send and receive events of a message labelled *req* from process p (*User1*) to process q (*Server*). Each pair of processes p and q is connected by a dedicated fifo channel (p, q)—for example, the messages sent at s_1 and s_2 are on channel (r, q) and the second message cannot be received before the first one.

Since processes are locally sequential, the set of events E_p along a process p is linearly ordered by a relation denoted $\leq_{pp}$. In addition, for each message sent along a channel (p, q), the send and receive events of the message are related by an ordering relation $\leq_{pq}$. Thus, for example, $a_1 \leq_{qq} a_5$ and $a_3 \leq_{qp} u_2$. Together, the local linear orders $\leq_{pp}$ and the message orders $\leq_{pq}$ generate a partial order $\leq$ over the set of events—for instance, $u_3 \leq s_3$.

Finally, we label each event using a finite alphabet *Act* of communication actions. We write $p!q(m)$ to denote the action where p sends message m to q and $p?q(m)$ to denote the action where p receives message m from q. We abbreviate by $p!q$ and $p?q$ the set of all actions of the form $p!q(m)$ and $p?q(m)$, respectively, over all possible choices of m.

Overall, an MSC can then be captured as a labelled partial order $M = (E, \leq, \lambda)$ where $\lambda : E \to Act$ associates each event with its corresponding action. A *cut* is a subset of events that is downward closed: $c \subseteq E$ is a cut if $\downarrow c = c$, where $\downarrow c = \{e \in E \mid \exists e' \in c.\ e \leq e'\}$.

Like any partial order, an MSC can be reconstructed upto isomorphism from its linearisations, i.e., words over *Act* that extend $\leq$. In fact, the fifo condition on channels ensures that a single linearisation suffices to reconstruct an MSC. In this way, an MSC M corresponds to a set $\mathrm{lin}(M)$ of words over *Act* and a set $\mathcal{L}$ of MSCs defines the word language $\bigcup_{M \in \mathcal{L}} \mathrm{lin}(M)$. We say that a set of MSCs $\mathcal{L}$ is *regular* if its associated word language is regular.

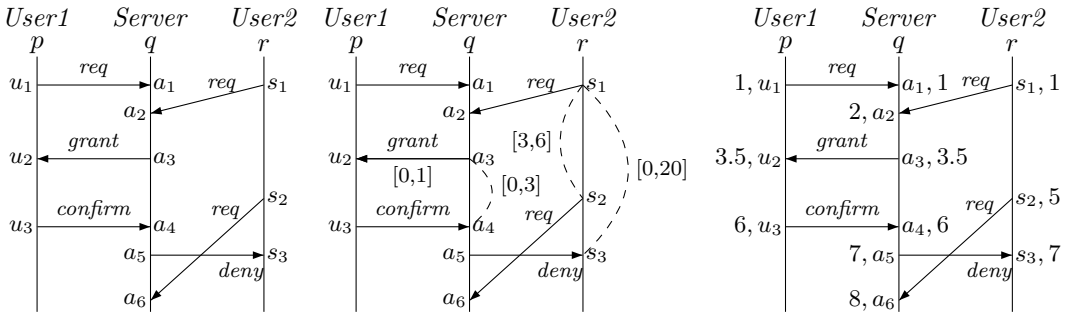

Figure 1 Different views of a system with two users with a server

2.2 Time-constrained message sequence charts

A *time-constrained MSC (TCMSC)* is an MSC annotated with time intervals between pairs of events. We restrict timing constraints to pairs of distinct events on the same process and to the matching send and receive events across messages. Intervals have rational endpoints and may be open or closed at either end.

For example, in the second diagram in Figure 1, the constraint $[0, 3]$ between a_3 and a_4 bounds the time that the *Server* waits for a *User* to confirm a grant. On the other hand, the constraint $[0, 1]$ between a_3 and u_2 bounds the time taken to deliver this particular message.

A TCMSC over Act is a pair $\mathfrak{M} = (M, \tau)$, where $M = (E, \leq, \lambda)$ is an MSC over Act and τ is a partial map from $E \times E$ to the set of intervals such that $(e, e') \in \mathrm{dom}(\tau)$ implies that $e \neq e'$ and either $e \leq_{pp} e'$ or $e \leq_{pq} e'$ for some processes p and q.

2.3 Timed message sequence charts

A *timed MSC (TMSC)* describes a concrete timed behaviour in the MSC setting. In a TMSC, we assign events timestamps that are consistent with the underlying partial order. Thus, a TMSC over Act is a pair $T = (M, t)$ where $M = (E, \leq, \lambda)$ is an MSC over Act and $t : E \to \mathbb{R}_{\geq 0}$ is a function such that if $e \leq e'$ then $t(e) \leq t(e')$ for all $e, e' \in E$.

For instance, consider the TMSC in the third diagram of Figure 1. The message sent at a_3 is received instantaneously while the message sent at s_2 is received 3 time units later.

A timed word over Act is a sequence $(a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ where $a_1 a_2 \cdots a_n$ is a word over Act and $t_1 \leq t_2 \leq \cdots \leq t_n$ is a nondecreasing sequence over $\mathbb{R}_{\geq 0}$. The set of timed words over Act is denoted TW_{Act}. A *timed linearisation* of a TMSC is thus a timed word in TW_{Act}. We let t-lin(T) denote the set of timed linearisations of TMSC T. A single TMSC may admit more than one timed linearisation if concurrent events on different processes have the same timestamp. As for untimed MSCs, under the fifo assumption for channels, a timed MSC can be reconstructed from any one of its timed linearisations.

With this definition, TCMSCs can be considered as abstractions of TMSCs and timed words. For instance, we will say that the TMSC in Figure 1 *realises* the TCMSC in the same figure since each interval constraint between events in the TCMSC is satisfied by the time-stamps of the corresponding events in the TMSC. In this way, a TCMSC $\mathfrak{M}$ defines a family of TMSCS—the set of all TMSCs that realise $\mathfrak{M}$, which we denote $\mathcal{L}_{time}(\mathfrak{M})$. We also consider the set $\mathcal{L}_{tw}(\mathfrak{M}) = \bigcup_{T \in \mathcal{L}_{time}(\mathfrak{M})} \text{t-lin}(T)$ of timed words that realise $\mathfrak{M}$.

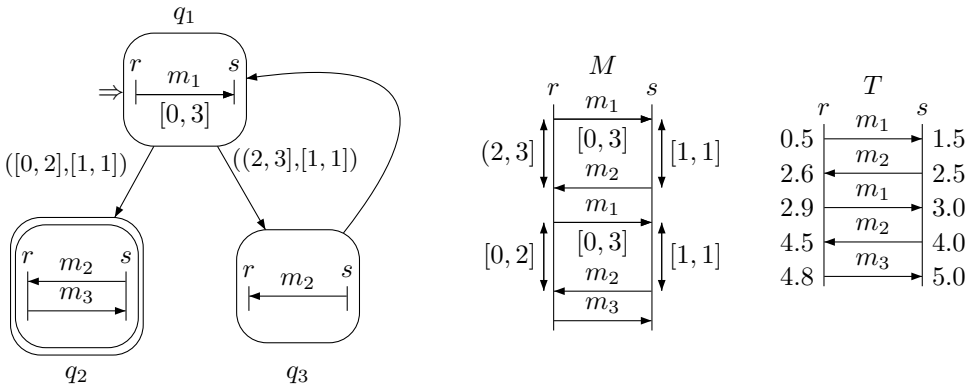

Figure 2 A TCMSG, with a TCMSC and a TMSC that it generates

2.4 Message sequence graphs

A *message sequence graph (MSG)* is a directed graph in which nodes are labelled by MSCs. We begin with a graph $G = (V, \to, v_{in}, V_F)$ with nodes V, initial node $v_{in} \in V$, final nodes $V_F \subseteq V$ and edge relation $\to$. An MSG is a structure $\mathfrak{G} = (G, \mathcal{L}^M, \Phi)$ where $\mathcal{L}^M$ is a set of basic MSCs and $\Phi : V \to \mathcal{L}^M$ associates a basic MSC with each node. An accepting path in G is a sequence of nodes $v_0 v_1 \cdots v_n$ that starts in v_{in} and ends in some node of V_F where each adjacent pair of states is related by $\to$. This path defines an MSC $\Phi(v_0 v_1 \cdots v_n) = \Phi(v_0) \circ \Phi(v_1) \circ \cdots \circ \Phi(v_n)$, where $\circ$ denotes MSC concatenation. When we concatenate two MSCs $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$ we attach the lifelines in M_2 below those of M_1 to obtain an MSC $M_1 \circ M_2 = (E^1 \cup E^2, \leq, \lambda)$ where λ combines λ_1 and λ_2 and $\leq$ is generated by $\leq^1 \cup \leq^2 \cup \{(e_1, e_2) \mid \exists p.\ e_1 \in E^1_p, e_2 \in E^2_p\}$.

Since each accepting path in an MSG defines an MSC, we can associate with an MSG $\mathfrak{G}$ a language $\mathcal{L}(\mathfrak{G})$ of MSCs. In general, it is undecidable to determine whether $\mathcal{L}(\mathfrak{G})$ is regular [7]. This is because processes move asynchronously along the MSC traced out by accepting paths and there is no bound, in general on this asynchrony. However, there is a sufficient structural condition to guarantee regularity [3, 10].

Given an MSC M, we construct its communication graph $CG(M)$ as follows: the vertices are the processes and we have a directed edge (p, q) if M contains a message from p to q. An MSC M is said to be *connected* if the non-isolated vertices in $CG(M)$ form a single strongly connected component. An MSG $\mathfrak{G}$ is said to be *locally synchronized* if for every loop π in $\mathfrak{G}$, the MSC $\Phi(\pi)$ is connected. Intuitively, this means that every message sent in a loop is implicitly acknowledged, because if p sends a message, there is a path in the communication graph back to p. This ensures that all channels are *universally bounded*—there is a uniform bound B such that across all linearisations, no channel ever has more than B pending messages. Thus, if $\mathfrak{G}$ is locally synchronized, $\mathcal{L}(\mathfrak{G})$ is a regular set of MSCs.

2.5 Time-constrained message sequence graphs

We generalise MSGs to the timed setting in a natural way. In a *time-constrained MSG (TCMSG)*, states are labelled by TCMSCs rather than basic MSCs. In addition, we also permit process-wise timing constraints along the edges of the graph. A constraint for process p along an edge $v \to v'$ specifies a constraint between the final p-event of $\Phi(v)$ and the initial p-event of $\Phi(v')$, provided p actively participates in both these nodes. If p does not participate in either of these nodes, the constraint is ignored. Formally, a TCMSG is a tuple $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ where $G = (V, \to, v_{in}, V_F)$ is a graph as before, $\Phi : V \to \mathcal{L}^{TC}$

labels each node with a TCMSC from a set $\mathcal{L}^{TC}$ and *EdgeC* associates a tuple of constraints with each edge—for convenience, we assume that any edge constraint not explicitly specified corresponds to the trivial constraint $(-\infty, \infty)$.

Each accepting path in a TCMSG defines a TCMSC. Given a path $v_0 v_1 \cdots v_n$, we concatenate the TCMSCs $\Phi(v_0), \Phi(v_1), \ldots, \Phi(v_n)$ and insert the additional constraints specified by *EdgeC*. We define $\mathcal{L}_{TC}(\mathfrak{G})$ to be the set of all TCMSCs over *Act* generated by accepting paths in G. We also let $\mathcal{L}_{time}(\mathfrak{G}) = \bigcup_{\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})} \mathcal{L}_{time}(\mathfrak{M})$ and $\mathcal{L}_{tw}(\mathfrak{G}) = \bigcup_{\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})} \mathcal{L}_{tw}(\mathfrak{M})$. Figure 2 shows a TCMSG, a TCMSC that it generates and a realizing TMSC.

2.6 Timed automata

We can formulate many types of machine models for timed MSCs. One natural choice is a message-passing automaton (MPA) equipped with (local) clocks. In a timed MPA, we have one component for each process p, which is a finite state automaton over actions of the form $p!q(m)$ and $p?q(m)$. Each component also has local clocks that can be used to guard transitions. The global state space defines a timed automaton over *Act*.

A timed automaton over an alphabet Σ is a tuple $\mathcal{A} = (Q, \Delta, q_{in}, F, Z)$ where Q is a finite set of states, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ are the final states and Z is a set of clocks that take values over $\mathbb{R}_{\geq 0}$. Each transition in Δ is of the form $q \xrightarrow{\varphi, a, X} q'$ where $q, q' \in Q$, $a \in \Sigma$, $X \subseteq Z$ and φ is a boolean combination of clock constraints of the form $x \ op \ c$ where $x \in Z$, $c \in \mathbb{Q}_{\geq 0}$ and $op \in \{\leq, <, >, \geq\}$. This transition is enabled if the current values of all clocks satisfy the guard φ. On taking this transition, the clocks in X are reset to 0. As is standard, time elapses between transitions, transitions occur instantaneously and such an automaton accepts timed words from TW_Σ. More details can be found in [2, 4].

For our purposes, we only need the following two results about timed automata.

- Given timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we can construct a timed automaton $\mathcal{A}_{12}$ such that $L(\mathcal{A}_{12}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
- Checking whether the language of a timed automaton is empty is decidable.

2.7 The model checking problem

We are interested in timed automata over *Act* whose languages can be interpreted as timed MSCs. A timed word in TW_{Act} corresponds to a linearisation of a timed MSC provided the timed word is well-formed and complete. A word w over *Act* is *well-formed* if for each channel (p, q), in every prefix v of w, the sequence of messages received by q from p in v is a prefix of the messages sent from p to q in v. A well-formed word w is *complete* if $\#_{p!q}(w) = \#_{q?p}(w)$ for each matching pair of send-receive actions, where $\#_X(u)$ counts the number of occurrences in u of $X \subseteq Act$. Finally, a well-formed word w is B-bounded if, in every prefix v of w, $\#_{p!q}(v) - \#_{q?p}(v) \leq B$ for each channel (p, q). Correspondingly, a timed word is said to be well-formed (complete, B-bounded) if its projection onto *Act* is well-formed (complete, B-bounded). Well-formedness captures the intuition that any receive action has an earlier matching sending action. Completeness guarantees that all pending messages have been received. B-boundedness promises that no channel ever has more than B messages.

Given a timed automaton $\mathcal{A}$ over *Act* and a TCMSG specification $\mathfrak{G}$, the model checking problem is to check that every timed word accepted by $\mathcal{A}$ realises some TCMSC in $\mathcal{L}_{TC}(\mathfrak{G})$. Since $\mathcal{A}$ may accept timed words that are not well-formed or not complete, this implicitly includes checking that $\mathcal{A}$ accepts only well-formed and complete timed words in TW_{Act}.

From this, it is clear that the model checking problem corresponds to checking whether $L(\mathcal{A}) \subseteq \mathcal{L}_{tw}(\mathfrak{G})$. To make the problem tractable, we restrict our attention to locally synchronized TCMSGs, so that $\mathcal{L}_{tw}(\mathfrak{G})$ is a timed regular language. Unfortunately, checking inclusion is undecidable even for timed regular languages [2]. To get around this, we introduce a more restricted machine model for timed MSCs called MSC event clock automata, which are closed under complementation. It turns out that $\mathcal{L}_{tw}(\mathfrak{G})$ can be recognized by MSC event clock automata, yielding a solution to our model checking problem.

3 An extended event clock automaton – the MSC-ECA

We now define MSC event clock automata or MSC-ECA. These will be used to capture exactly the guards that occur in the TCMSGs that we have defined. We denote an MSC-ECA over Act by $\mathcal{C} = (Q, Act, \delta, q_0, F)$, with states Q, initial state $q_0 \in Q$ and final states $F \subseteq Q$. A transition in δ is of the form (q, φ, a, q') where $q, q' \in Q, a \in Act$ and φ is a conjunction of event clock guards, which are of two types: either $\mathrm{Y}_p^k \in I$ or $\mathrm{Msg}^{-1} \in I$, where I is an interval, as used in TCMSC timing constraints. We interpret these guards over timed words. Let $\sigma = (a_1, t_1) \cdots (a_n, t_n) \in \mathrm{TW}_{Act}$. Then at a position $1 \leq j \leq n$, we define

(D1) $\sigma, j \models \mathrm{Y}_p^k \in I$ if the time elapsed between the k^{th}-previous p-action a_i in σ and this action a_j is in the interval I.

(D2) $\sigma, j \models \mathrm{Msg}^{-1} \in I$ if a_j is a receive action and the time elapsed since the occurence of its matching send action a_i is in the interval I.

In both these definitions, note that action a_i is uniquely defined, i.e., there is at most one position i that matches a given position j with respect to a given event clock guard.

Now, we define runs of $\mathcal{C}$ over timed words. For a timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$, we say there is a run of $\mathcal{C}$ from q to q' on σ, denoted $q \xrightarrow{\sigma} q'$ *in* $\mathcal{C}$, if there exists a sequence of transitions $q = q_0 \xrightarrow{\varphi_1, a_1} \cdots \xrightarrow{\varphi_n, a_n} q_n$ such that for all j, $1 \leq j \leq n$, $\sigma, j \models \varphi_j$. The timed word σ is said to be accepted if it has a run from the initial to some final state in F. We denote by $\mathcal{L}_{tw}(\mathcal{C})$ the set of timed words accepted by the MSC-ECA $\mathcal{C}$.

3.1 Determinization and complementation of MSC-ECA

We now prove that MSC-ECA can be determinized and complemented, which is crucial for solving the model checking problem. We obtain this by constructing a deterministic and complete version of any given MSC-ECA. Intuitively, this works as for classical ECA's and the main reason is that there are no explicit clocks. Since the reset of an event clock only depends on the timed word being read and not on the path followed in the automaton, we can use the subset construction.

More precisely, let $\mathcal{C} = (Q, Act, \delta, q_0, F)$ be a finite MSC-ECA. The set of states of the universal automaton $\mathcal{C}^{univ}$ is 2^Q. For a set $X \subseteq Q$ and an action a, we let $T(X, a)$ denote the set of transitions in δ having action a and a source state in X. Then, for some $T' \subseteq T(X, a) = T$, we denote by $\mathrm{target}(T')$ the set of target states of transitions in T' and we define

$$\varphi(T', T) = \bigwedge_{t=(q,\varphi_t,a,q') \in T'} \varphi_t \wedge \bigwedge_{t=(q,\varphi_t,a,q') \in T \setminus T'} \neg \varphi_t \ .$$

Then, we denote the set of transitions of $\mathcal{C}^{univ}$ by Δ, where we say that $X \xrightarrow{\varphi, a} X' \in \Delta$ if there exists $T' \subseteq T = T(X, a)$ such that $\varphi = \varphi(T', T)$ and $X' = \mathrm{target}(T')$.

Note that, once we have fixed X, a and the set T', the transition is uniquely defined. Also for $X = \emptyset$, we have $T(X, a) = \emptyset$ and the only possible transition is $\emptyset \xrightarrow{\text{true},a} \emptyset$. The crucial property of $\mathcal{C}^{univ}$ is that it is deterministic and complete (and finite, if $\mathcal{C}$ is).

▶ **Lemma 1.** *Given any timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n) \in \mathrm{TW}_{Act}$, there exists a unique run $X_0 \xrightarrow{\varphi_1,a_1} X_1 \xrightarrow{\varphi_2,a_2} \cdots X_{n-1} \xrightarrow{\varphi_n,a_n} X_n$ of $\mathcal{C}^{univ}$ on σ starting from $X_0 = \{q_0\}$. Moreover, $X_n = \{q \in Q \mid q_0 \xrightarrow{\sigma} q \text{ in } \mathcal{C}\}$.*

By suitably choosing the final states, $\mathcal{C}^{univ}$ will accept either the same language as $\mathcal{C}$ or its complement. Let $F_1 = \{X \in 2^Q \mid F \cap X \neq \emptyset\}$ and $F_2 = 2^Q \setminus F_1$. Define $\mathcal{C}_i^{univ} = (2^Q, Act, \Delta, \{q_0\}, F_i)$ for $i = \{1, 2\}$. From Lemma 1 we obtain:

▶ **Corollary 2.** *We have $\mathcal{L}_{tw}(\mathcal{C}_1^{univ}) = \mathcal{L}_{tw}(\mathcal{C})$ and $\mathcal{L}_{tw}(\mathcal{C}_2^{univ}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathcal{C})$.*

3.2 From MSC-ECA to TA

Not every MSC-ECA can be translated into an equivalent (classical) timed automaton. The problem comes from the event guards $\mathrm{Msg}^{-1} \in I$, which may require infinitely many clocks if channels are unbounded. Fortunately, thanks to the locally synchronized assumption on TCMSGs, we are only interested in bounded channels. Let $B > 0$. We show below how to construct a timed automaton $\mathcal{B}_{\mathcal{C}}^B$ from an MSC-ECA $\mathcal{C} = (Q, Act, \delta, q_0, F)$ such that $\mathcal{B}_{\mathcal{C}}^B$ and $\mathcal{C}$ are equivalent when restricted to B-bounded channels.

Let $K = \max\{k \mid \mathrm{Y}_p^k \in I \text{ occurs in some guard in } \delta\}$. A state of $\mathcal{B}_{\mathcal{C}}^B$ is either a dead state denoted $\perp$ or a tuple $\mathfrak{s} = (s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$ where $s \in Q$, $\overline{b} = (b_p)_{p \in Proc} \in \{0, 1\}^{Proc}$ ($b_p = 1$ if we have already seen at least K p-events), $\overline{n} = (n_p)_{p \in Proc} \in \{0, \ldots, K-1\}^{Proc}$ (n_p is the number of p-events already seen modulo K), $\overline{\alpha} = (\alpha_{p,q})_{p,q \in Proc} \in \{0, \ldots, B\}^{Proc^2}$ ($\alpha_{p,q}$ is the number of $q?p$ events modulo $B+1$), $\overline{\beta} = (\beta_{p,q})_{p,q \in Proc} \in \{0, \ldots B\}^{Proc^2}$ ($\beta_{p,q}$ is the number of $p!q$ events modulo $B+1$). The set of all states is denoted Q' and the initial state is $\mathfrak{s}_0 = (s_0, (0), (0), (0), (0))$. The set of clocks is $Y \cup Z$ where $Y = \{y_p^i \mid p \in Proc, 0 \leq i < K\}$ and $Z = \{z_{p,q}^i \mid p, q \in Proc, 0 \leq i \leq B\}$. We will reset clock y_p^i when executing the i^{th} p-event mod K. Also, $z_{p,q}^i$ will be reset when executing the i^{th} $p!q$ event mod $B+1$.

We say that channel (p, q) is *empty* if $\alpha_{p,q} = \beta_{p,q}$ and *full* if $\beta_{p,q} = \alpha_{p,q} + B \bmod (B+1)$. The set of transitions $\delta_{\mathcal{B}_{\mathcal{C}}^B}$ is defined as follows: Assume $s \xrightarrow{\varphi,a} s'$ in $\mathcal{C}$ with $a \in Act_p$. Then, we have three types of transitions in $\mathcal{B}_{\mathcal{C}}^B$. (Recall that $p!q$ and $p?q$ abbreviate all actions of the form $p!q(m)$ and $p?q(m)$, respectively.)

(Tr1) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\text{true},a,\emptyset} \perp$ is in $\mathcal{B}_{\mathcal{C}}^B$ if either $a \in p!q$ and channel (p, q) is full (the bound was exceeded), or $a \in p?q$ and channel (p, q) is empty.

(Tr2) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\varphi',a,R} (s', \overline{b}', \overline{n}', \overline{\alpha}', \overline{\beta}')$ is in $\mathcal{B}_{\mathcal{C}}^B$ if we are not in the above case and the following conditions hold:

1. $b'_p = 1$ if $n_p = K - 1$ and $b'_p = b_p$ otherwise. Also, $b'_r = b_r$ for $r \neq p$.
2. $n'_p = (n_p + 1) \bmod K$ and $n'_r = n_r$ for $r \neq p$.
3. if $a \in p!q$, then $\beta'_{p,q} = (\beta_{p,q} + 1) \bmod (B+1)$ and $\beta'_{p',q'} = \beta_{p',q'}$ for $(p', q') \neq (p, q)$.

Also $\overline{\alpha}' = \overline{\alpha}$, $R = \{y_p^{n'_p}, z_{p,q}^{\beta'_{p,q}}\}$ and φ' is φ where $\mathrm{Y}_p^k \in I$ is replaced with

$$\begin{cases} \text{false} & \text{if } b_p = 0 \text{ and } k > n_p \\ y_p^{(K+n'_p-k) \bmod K} \in I & \text{otherwise} \end{cases}$$

4. if $a \in p?q$, then $\alpha'_{q,p} = \alpha_{q,p} + 1 \bmod (B+1)$ and $\alpha'_{q',p'} = \alpha_{q',p'}$ for $(q',p') \neq (q,p)$. Also $\overline{\beta}' = \overline{\beta}$, $R = \{y_p^{n'_p}\}$ and φ' is φ where $\mathrm{Y}_p^k \in I$ is replaced as above and $\mathrm{Msg}^{-1} \in I$ is replaced with $z_{q,p}^{\alpha'_{q,p}} \in I$.

(Tr3) $\bot \xrightarrow{\mathrm{true},a,\emptyset} \bot$ is in $\mathcal{B}_{\mathcal{C}}^B$ for all $a \in Act$.

In the following, we call a timed word w *weakly well-formed* (wwf) if for each channel (p,q), in every prefix v of w, we have $\#_{q?p}(w) \leq \#_{p!q}(w)$. This weak form does not require the send message sequence to be the same as the received one. Let $\mathrm{TW}_{Act}^{B,\mathrm{wf}}$ denote the set of timed words $\sigma \in \mathrm{TW}_{Act}$ which are both wwf and B-bounded. We can define different notions of acceptance (i.e., final states) on $\mathcal{B}_{\mathcal{C}}^B$ constructed from $\mathcal{C}$ to derive the results below.

▶ **Proposition 3.** *Let $\mathcal{C} = (Q, Act, \delta, q_0, F)$ and $\mathcal{B}_{\mathcal{C}}^B = (Q', Act, (Y \cup Z), \delta_{\mathcal{B}_{\mathcal{C}}^B})$ be as above.*

1. *With final states $F' = \{(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \mid s \in F\}$ the timed automaton $\mathcal{B}_{\mathcal{C}}^B$ accepts the language $\mathcal{L}_{tw}(\mathcal{C}) \cap \mathrm{TW}_{Act}^{B,\mathrm{wf}}$.*
2. *If $\mathcal{C}$ is complete (i.e., it has a run on every timed word over Act) then with final states $F'' = \{\bot\}$ the timed automaton $\mathcal{B}_{\mathcal{C}}^B$ accepts the complement of $\mathrm{TW}_{Act}^{B,\mathrm{wf}}$.*

Proof. (Sketch) Let $\sigma = (a_1,t_1)\cdots(a_m,t_m)$ be a wwf and B-bounded timed word. Consider a path $\pi = s_0 \xrightarrow{\varphi_1,a_1} s_1 \xrightarrow{\varphi_2,a_2} \cdots \xrightarrow{\varphi_m,a_m} s_m$ of $\mathcal{C}$. We can build inductively a path $\pi' = \mathfrak{s}_0 \xrightarrow{\varphi'_1,a_1,R_1} \mathfrak{s}_1 \xrightarrow{\varphi'_2,a_2,R_2} \cdots \xrightarrow{\varphi'_m,a_m,R_m} \mathfrak{s}_m$ of $\mathcal{B}_{\mathcal{C}}^B$ starting from its initial state $\mathfrak{s}_0$ and using *(Tr2)* only. Then, we can prove that if σ has a run through π in $\mathcal{C}$ (i.e., $\sigma, i \models \varphi_i$ for all $i \in \{1,\ldots,m\}$) then σ has a run through π' in $\mathcal{B}_{\mathcal{C}}^B$. Hence we obtain one inclusion of (1).

For the converse inclusion, we start with a path of $\mathcal{B}_{\mathcal{C}}^B$ starting from its initial state $\mathfrak{s}_0$ and which does not reach $\bot$: $\pi' = \mathfrak{s}_0 \xrightarrow{\varphi'_1,a_1,R_1} \mathfrak{s}_1 \xrightarrow{\varphi'_2,a_2,R_2} \cdots \xrightarrow{\varphi'_m,a_m,R_m} \mathfrak{s}_m$. Since we did not reach $\bot$, the timed word $\sigma = (a_1,t_1)\cdots(a_m,t_m)$ must be wwf and B-bounded. Moreover, transitions in π' comes from *(Tr2)* only and we can recover a corresponding path $\pi = s_0 \xrightarrow{\varphi_1,a_1} s_1 \xrightarrow{\varphi_2,a_2} \cdots \xrightarrow{\varphi_m,a_m} s_m$ in $\mathcal{C}$. Again, we can prove that if σ has a run through π' in $\mathcal{B}_{\mathcal{C}}^B$ then σ has a run through π in $\mathcal{C}$.

Statement (2) can be proved easily. ◀

4 From a locally synchronized TCMSG to a finite MSC-ECA

The main result is that locally synchronized TCMSGs define timed regular languages.

▶ **Theorem 4.** *If $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ is a locally synchronized TCMSG, then there exists a finite MSC-ECA $\mathcal{C}$, such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$.*

In the untimed case, the corresponding result has been stated and proved in different ways [3, 5, 6, 10]. We describe a different proof that is more suitable for the timed version.

We want to simulate the global run of a TCMSG by keeping a finite amount of information in the states of the MSC-ECA. Intuitively, we keep the sequence of nodes along the TCMSG path that have been started but not completed (at least one executed event but not all). Since the TCMSG is locally synchronized, the number of such nodes is always bounded.

We replace segments of nodes in the TCMSG path that have not been started yet by a special gap symbol #. Nodes will be inserted at gaps whenever necessary, making sure that the sequential run of the MSC-ECA is compatible with the TCMSG path. In fact, the insertion must satisfy two conditions: (1) when we insert a node it must not conflict with the events that have already occurred in later nodes and (2) finally, after all insertions, we

do obtain a path in the MSG. The latter is done by checking that when we fill a gap the corresponding bordering nodes have an edge in the graph.

We also replace segments of fully executed nodes of a TCMSG path by the set of processes that have been active in these nodes, so that we ensure condition (1) above.

For a node u, let E^u be the set of events in the MSC labelling u. We define an *extended node* to be a pair (u, c) where $u \in V$ and $c \subseteq E^u$ is a cut of E^u that contains the events that have been executed in node u. For simplicity, we extend the set of vertices V with two dummy vertices $\triangleright$, $\triangleleft$ and add edges from $\triangleright$ to the initial vertex v_{in} and from every final vertex $v \in V_F$ to $\triangleleft$. We also set $E^{\triangleright} = \emptyset = E^{\triangleleft}$ so that for $u \in \{\triangleright, \triangleleft\}$, the only extended node is $(u, \emptyset)$. The set of all extended nodes is denoted *ExtNodes*. An extended node (u, c) is said to be *completed* if $c = E^u$. Note that $(\triangleright, \emptyset)$ and $(\triangleleft, \emptyset)$ are completed by default.

A state α of our new automaton $\mathcal{C}$ is a sequence of extended nodes, gaps and subsets of processes: $\alpha \in \Pi^*$ where $\Pi = \mathit{ExtNodes} \uplus \{\#\} \uplus 2^{\mathit{Proc}}$. The *initial* state is $\alpha_0 = (\triangleright, \emptyset)\#(\triangleleft, \emptyset)$. *Final* states are of the form $(\triangleright, \emptyset)P(\triangleleft, \emptyset)$ where $P \subseteq \mathit{Proc}$.

An *extended event* of $\alpha \in \Pi^*$ is a pair $(e, \alpha_1(u, c))$ where $e \in E^u$ and $\alpha_1(u, c) \preceq \alpha$—i.e., $\alpha_1(u, c)$ is a prefix of α. We say that the extended event $(e, \alpha_1(u, c))$ is *executed* in α if $e \in c$ and *enabled* in α if the following hold:

(E1) It has not been executed, i.e., $e \notin c$.
(E2) All events within the node which are below it (in the partial order) have been executed, i.e., for all $e' \in E^u$ with $e' <^u e$, we have $e' \in c$.
(E3) If e belongs to process p, then all p-events on any node occurring before this node in α have been executed, i.e., if $e \in E^u_p$ then for all $\alpha'_1(u', c') \preceq \alpha_1$, we have $E^{u'}_p \subseteq c'$.

We introduce notation to describe the set of processes that participate in nodes, paths or states. For a node $u \in V$, $\mathit{OProc}(u) = \{p \in \mathit{Proc} \mid E^u_p \neq \emptyset\}$ denotes the set of processes that participate (*occur*) in u. This is extended to V^* as a morphism. Also, with $\mathit{OProc}((u, c)) = \mathit{OProc}(u)$, $\mathit{OProc}(\#) = \emptyset$ and $\mathit{OProc}(P) = P$, it extends to Π^*. In addition, for $\beta \in \Pi^*$, $\mathit{EProc}(\beta)$ denoting the set of processes having *executed* events in β, is given by the morphism defined by $\mathit{EProc}((u, c)) = \{p \in \mathit{Proc} \mid E^u_p \cap c \neq \emptyset\}$, $\mathit{EProc}(\#) = \emptyset$ and $\mathit{EProc}(P) = P$.

Now, the transitions can be defined by saying that at any state α we can choose to execute an enabled (extended) event or add a new (extended) node in a gap of the state, in which case we must execute an enabled event on the new node.

We first define the *node insertion* operation as a macro $\alpha_1\#\alpha_2 \xrightarrow{u} \alpha'_1(u, \emptyset)\alpha'_2$ which is said to hold if

(I1) for every process that participates in u, there is no executed event in the segment α_2 on that process, i.e., $\mathit{OProc}(u) \cap \mathit{EProc}(\alpha_2) = \emptyset$.
(I2) $\alpha'_1 \in \{\alpha_1, \alpha_1\#\}$ and if $\alpha'_1 = \alpha_1$ then $\alpha_1 = \alpha''_1(v, c)$ and $v \to u$ in G.
(I3) $\alpha'_2 \in \{\alpha_2, \#\alpha_2\}$ and if $\alpha'_2 = \alpha_2$ then $\alpha_2 = (v, c)\alpha''_2$ and $u \to v$ in G.

Next, we explain how completed nodes are deleted from a state α. To check (I1) we need to preserve the set of executed processes, hence a completed node u will be replaced by $\mathit{OProc}(u)$. We also preserve (do not throw away) the nodes around a gap in α so that conditions (I2)–(I3) can still be checked. Finally, we preserve nodes that start an edge constraint which needs to be verified later (this is useful for guards defined in the transition relation below). Formally, we define the reduction as a rewrite operation $\alpha \xrightarrow{\mathit{redn}} \alpha'$. There are two rewrite rules:

(R1) $\alpha_1 P P' \alpha_2 \xrightarrow{\mathit{redn}} \alpha_1(P \cup P')\alpha_2$, i.e., two adjacent process sets can be merged.

(R2) $\alpha_1(v, E^v)\alpha_2 \xrightarrow{redn} \alpha_1 OProc(v)\alpha_2$ (a completed node is replaced by the set of processes participating in it) if the following hold:

(C2.1) $v \in V$, $\varepsilon \neq \alpha_1 \notin \Pi^*\#$, $\varepsilon \neq \alpha_2 \notin \#\Pi^*$ i.e., the node v is not next to a gap or at the beginning or the end.

(C2.2) (i) either the first symbol of α_2 is an extended node (v', c') and if both E^v_p and $E^{v'}_p$ are nonempty, then some event in $E^{v'}_p$ has occured (hence the edge constraint, if any, has already been checked),

(ii) or $\alpha_2 \in 2^{Proc}\Pi^*$ in which case there is no unchecked edge constraint.

▶ **Lemma 5.** *The rewrite system defined by the operation* $\xrightarrow{redn}$ *is confluent.*

Using the above lemma we conclude that, from any state α, after any maximal sequence of reductions, we reach the same state, which we denote by $Red(\alpha)$.

Now, we can define the transition relation: $\alpha \xrightarrow{\varphi,a} \alpha'$ is a transition in $\mathcal{C}$ if there exists $\beta = \beta_1(u, c)\beta_2$ and an extended event $(e, \beta_1(u, c))$ enabled in β such that

(i) either $\beta = \alpha$, i.e., the enabled event is already present in the current state,

(ii) or $\alpha = \alpha_1\#\alpha_2 \xrightarrow{u} \beta_1(u, \emptyset)\beta_2 = \beta$. Hence, $c = \emptyset$, $\beta_1 \in \{\alpha_1, \alpha_1\#\}$ and $\beta_2 \in \{\alpha_2, \#\alpha_2\}$

and all the following conditions hold:

(T1) $a = \lambda^u(e)$.

(T2) The guard φ checks all local and edge constraints—i.e.,

$$\varphi = \left(\bigwedge_{e' \in E^u, I \in \mathcal{I} \mid \tau^u(e', e) = I} \varphi(u, e', e, I) \right) \wedge \varphi^{edge} \text{ where,} \tag{1}$$

$$\varphi(u, e', e, I) = \begin{cases} \mathrm{Msg}^{-1} \in I & \text{if } \exists p, q, p \neq q \text{ s.t. } e' <^u_{qp} e \\ \mathrm{Y}^k_p \in I & \text{if } e, e' \in E^u_p \text{ and } |\{e'' \in E^u_p \mid e' \leq^u_{pp} e'' <^u_{pp} e\}| = k \end{cases} \tag{2}$$

$$\text{and } \varphi^{edge} = \begin{cases} \mathrm{Y}^1_p \in I & \text{if } \beta_1 = \beta'_1(u', c'') \text{ and for some } p \in Proc, \text{ we have} \\ & EdgeC((u', u), p) = I \text{ and } e = min(E^u_p) \\ \text{true} & \text{otherwise} \end{cases} \tag{3}$$

(T3) $\alpha' = Red(\beta_1(u, c')\beta_2)$ where $c' = c \uplus \{e\}$.

Observe that, once the state and the enabled event which is to be executed are fixed, the transition that is taken and indeed the state reached after the transition are uniquely determined. We can also observe that every reachable state α of $\mathcal{C}$ is *valid*. By this we mean that it satisfies the following properties:

(V1) Every # symbol in α is surrounded by nodes from *ExtNodes*. Also α starts with $(\triangleright, \emptyset)$ and ends with $(\triangleleft, \emptyset)$.

(V2) For any two consecutive extended nodes in α, there exists an edge between the nodes in G, i.e., for all $\alpha_1(u, c)(u', c') \preceq \alpha$, we have $u \to u'$ in G.

(V3) Executed events in α are *downward closed*:

a. For all $\alpha_1(u, c) \preceq \alpha$, if $e \in c$ and $e' \leq^u e$ then $e' \in c$.

b. For all $\alpha_1(u, c)\alpha_2(u', c') \preceq \alpha$, if $e \in E^u_p$ and $e' \in c' \cap E^{u'}_p$ for some p, then $e \in c$.

In order to get finiteness of the automaton $\mathcal{C}$, we need to restrict to states that are both reachable and *completable*. Formally, we call a state α *completable* if whenever $\alpha = \alpha_1(u,c)\#(v,c')\alpha_2$, there is $\beta \in V^+$ such that $u\beta v$ is a path in G and $OProc(\beta) \cap EProc((v,c')\alpha_2) = \emptyset$. Note that, in order to be co-reachable in $\mathcal{C}$, a state must be completable.

▶ **Lemma 6.** *If $\mathfrak{G}$ is locally synchronized, the set of states of $\mathcal{C}$ which are both valid and completable is finite.*

Proof. (Sketch) It is enough to show that the length of each valid, completable state of $\alpha \in \Pi^*$ is bounded. By definition, every extended node in α has at least one executed event. Using the locally synchronized assumption, one can prove the following properties about a loop in a state.

▶ Claim 7. Let $\alpha(u,c)\beta(u,c')\gamma$ be a valid completable state of $\mathcal{C}^{fin}_{\mathfrak{G}}$. If $(u,c)\beta$ is not completely executed or if $\#$ occurs in β, then $EProc((u,c')\gamma) \subsetneq EProc((u,c)\beta(u,c')\gamma)$.

Now, consider a loop $\alpha(u,c)\beta(u,c')\gamma$ in a valid completable state. If β has no $\#$ and $(u,c)\beta$ is completely executed, then $\alpha = \alpha'\#$. Indeed, otherwise the completed node (u,c) would have been deleted. Along with the previous claim this implies that we can bound the number of occurences of a node u in a path by $2|Proc|$. From which we can conclude that we have a bound of $2|Proc||V|$ on the number of extended nodes in a path. But we know that each $\#$ or $P \subseteq Proc$ must have an extended node next to it on the left. So we can conclude that the length of the path is $\mathcal{O}(|Proc||V|)$. Thus $\mathcal{C}$ is finite. ◀

The main result is stated in the following proposition.

▶ **Proposition 8.** $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$.

The proof which is long and technical is omitted for lack of space. It can be found in [1] where it is split in three main steps. First we construct an MSC-ECA with infinitely many states: we guess the full path of the TCMSG initially and we keep it in all states along the run to avoid the complication of node insertions and node deletions. Next, we introduce the automaton with gaps, dealing with node insertions but not yet with node deletions. This automaton is still infinite. Finally we introduce node deletions to obtain the automaton $\mathcal{C}$ constructed above. At each step we prove the equality of the timed languages, either directly, or using bisimulation at the abstract level of paths.

5 Solving the model checking problem

Now, we are in a position to solve the model checking problem.

▶ **Theorem 9.** *For a locally synchronized TCMSG $\mathfrak{G}$ and a timed automaton $\mathcal{A}$, the model checking problem $\mathcal{L}_{tw}(\mathcal{A}) \subseteq \mathcal{L}_{tw}(\mathfrak{G})$ is decidable, i.e., it is decidable to check if for all timed words σ generated by $\mathcal{A}$ there exists some $\mathfrak{M}$ specified by $\mathfrak{G}$ such that σ is a linearisation of a TMSC T which realises $\mathfrak{M}$.*

Proof. We have to prove that $\mathcal{L}_{tw}(\mathcal{A}) \cap (\mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})) = \emptyset$. By Theorem 4 we can construct an MSC-ECA $\mathcal{C}$ such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$. Using the complementation construction of Section 3.1 we can build a deterministic and complete MSC-ECA $\mathcal{C}' = \mathcal{C}_2^{univ}$ such that by Corollary 2 we have $\mathcal{L}_{tw}(\mathcal{C}') = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathcal{C}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

Since $\mathfrak{G}$ is locally synchronized, there is a bound $B > 0$ such that each timed word $\sigma \in \mathcal{L}_{tw}(\mathfrak{G})$ is wwf and B-bounded: $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{Act}$. Consider the timed automaton $\mathcal{B}^B_{\mathcal{C}'}$

associated with $\mathcal{C}'$ and the bound B by the construction of Section 3.2. For final states of $\mathcal{B}^B_{\mathcal{C}'}$ we choose $F' \cup F''$ as defined in Proposition 3. We get $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = (\mathrm{TW}_{Act} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{Act}) \cup (\mathcal{L}_{tw}(\mathcal{C}') \cap \mathrm{TW}^{B,\mathrm{wf}}_{Act}) = (\mathrm{TW}_{Act} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{Act}) \cup (\mathrm{TW}^{B,\mathrm{wf}}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G}))$. Using $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{Act}$ we deduce $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

Hence, the model checking problem is reduced to checking emptiness of the intersection of two timed automata, $\mathcal{A}$ and $\mathcal{B}^B_{\mathcal{C}'}$, which is indeed decidable. ◀

References

1. S. Akshay, P. Gastin, M. Mukund and K. Narayan Kumar: Model checking time-constrained scenario-based specifications. Technical Report LSV-10-16, ENS Cachan, 2010. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rapports`.
2. R. Alur and D. Dill: A Theory of Timed Automata. *Theor. Comput. Sci.*, **126** (1994) 183–225.
3. R. Alur and M. Yannakakis: Model checking of message sequence charts. *Proc. CONCUR'99*, Springer LNCS **1664** (1999) 114–129
4. J. Bengtsson and Wang Yi: Timed Automata: Semantics, Algorithms and Tools, *Lectures on Concurrency and Petri Nets 2003*, Springer LNCS **3098** (2003) 87–124.
5. J. Chakraborty, D. D'Souza, and K. Narayan Kumar. Analysing message sequence graph specifications. Technical Report IISc-CSA-TR-2009-1, IISc Bangalore, 2009.
6. M. Clerbout and M. Latteux. Semi-commutations. *Inf. Comp.*, **73(1)** (1987) 59–74.
7. J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P.S. Thiagarajan: A Theory of Regular MSC Languages. *Inf. Comp.*, **202(1)** (2005) 1–38.
8. ITU-T Recommendation Z.120: *Message Sequence Chart (MSC)*. ITU, Geneva (1999).
9. S. Mauw and M.A. Reniers: High-level message sequence charts. *Proc. SDL'97*, Elsevier (1997) 291–306.
10. A. Muscholl and D. Peled: Message sequence graphs and decision problems on Mazurkiewicz traces. *Proc. MFCS'99*, Springer LNCS **1672** (1999) 81–91.