Model checking time-constrained scenario-based specifications

S. Akshay, Paul Gastin, Madhavan Mukund,
K. Narayan Kumar

October 2010

Research report LSV-10-16

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Model checking time-constrained scenario-based specifications[⋆]

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

[1] LSV, ENS Cachan, INRIA, CNRS, France
{akshay,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.

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.

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

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

languages, which is known to be undecidable even for timed regular languages [1]. 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

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 $\mathcal{M}$. 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 $\lambda : E \to \mathit{Act}$ 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 of MSCs $\mathcal{L}$ defines the word language $\bigcup_{M \in \mathcal{L}} \mathrm{lin}(M)$. We say that a set $\mathcal{L}$ of MSCs is *regular* if its associated word language is regular.

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.

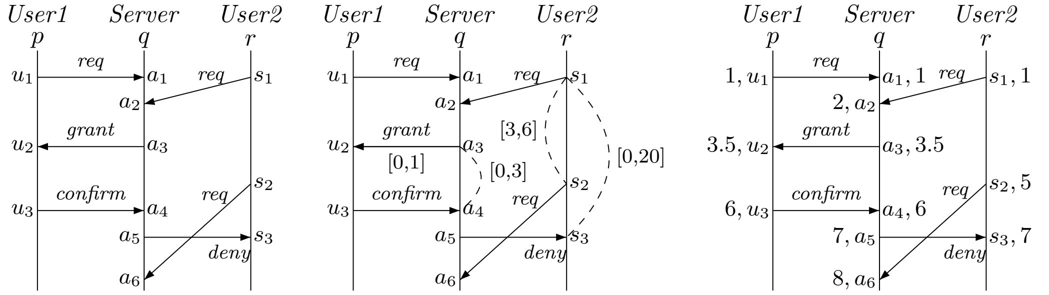

Fig. 1. Different views of a system with two users and a server

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.

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 timestamps 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 $\mathfrak{M}$.

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. A path in G is a sequence of nodes $v_0 v_1 \cdots v_n$ where each adjacent pair of states is related by $\to$. An accepting path is one that starts in v_{in} and ends in some node of V_F

A path $\pi = v_0 v_1 \cdots v_n$ in G 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 $M_1 = (E^1, \leq^1, \lambda_1)$

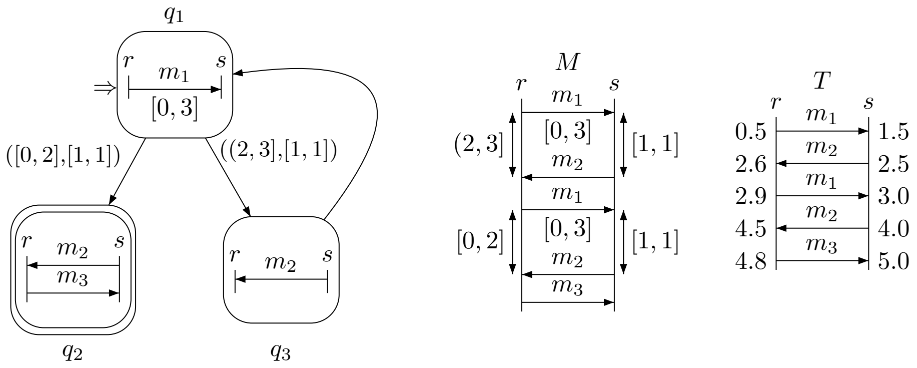

Fig. 2. A TCMSG, with a TCMSC and a TMSC that it generates

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 $\leq^1 \cup \leq^2 \cup \{(e_1, e_2) \mid \exists p.\ e_1 \in E^1_p, e_2 \in E^2_p\}$.

More formally, for a path $\pi = v_0 v_1 \cdots v_n$ we define the MSC $\Phi(\pi)$ as follows. For each vertex v , let $\Phi(v)$ be the MSC $M_v = (E^v, \leq^v, \lambda_v)$. We assume that the events are disjoint across the MSCs M_v. We then define $\Phi(\pi) = M_\pi = (E^\pi, \leq^\pi, \lambda_\pi)$, where,

- $E^\pi = \bigcup_{\rho v \preceq \pi} E_v \times \{\rho v\}$
- For each $\rho v \preceq \pi$, $\lambda_\pi(e, \rho v) = \lambda_v(e)$.
- $\leq^\pi$ is defined as the reflexive transitive closure of $\bigcup_{p,q \in Proc} <^\pi_{pq}$, where
  - $(e, \rho v) <^\pi_{pp} (e', \rho' v')$ for some $p \in Proc$ if $e \in E^v_p$, $e' \in E^{v'}_p$ and either $\rho v \preceq \rho' v'$ or ($\rho v = \rho' v'$ and $e <^v_{pp} e'$).
  - $(e, \rho v) <^\pi_{pq} (e', \rho' v')$ for some processes $p \neq q$, if $\rho v = \rho' v'$ and $e <^v_{pq} e'$.

We associate with an MSG $\mathfrak{G}$ a language of MSCs $\mathcal{L}(\mathfrak{G}) = \{\Phi(\pi) \mid \pi$ is an accepting path in $G\}$. 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.

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 $G = (V, \rightarrow, v_{in}, V_F)$ is a graph as before, $\Phi : V \rightarrow \mathcal{L}^{TC}$ 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 realising TMSC.

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 [1, 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.

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 [1]. 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'), which we also write as $q \xrightarrow{\varphi,a} q'$, 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. Formally, $a_j \in Act_p$ and there exists $1 \leq i < j$ such that $a_i \in Act_p$, $|\{\ell \mid i \leq \ell < j \wedge a_\ell \in Act_p\}| = k$ and $t_j - t_i \in I$.

(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. Formally, if there exists $p, q \in Proc$, $m \in \mathcal{M}$, $1 \leq i < j$ such that $a_i = p!q(m), a_j = q?p(m)$, $|\{a_k \mid 1 \leq k \leq i, a_k \in p!q\}| = |\{a_k \mid 1 \leq k \leq j, a_k \in q?p\}|$ and $t_j - t_i \in I$ (recall that we write $a_k \in p!q$ and $a_k \in q?p$ to mean $a_k = p!q(m)$ and $a_k = q?p(m)$ for some $m \in \mathcal{M}$, respectively).

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 the MSC-ECA $\mathcal{C}$ 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 state to some final state in F. We denote by $\mathcal{L}_{tw}(\mathcal{C})$ the set of timed words accepted by the MSC-ECA $\mathcal{C}$. An MSC-ECA is said to be *finite* if it has finitely many states.

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

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

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 $\emptyset \xrightarrow{\mathrm{true},a} \emptyset$. As before, a run on a timed word $\sigma = (a_1,t_1)\cdots(a_n,t_n)$ is defined as a sequence of transitions $X_0 \xrightarrow{\varphi_1,a_1} X_1 \cdots \xrightarrow{\varphi_n,a_n} X_n$ such that $\sigma, j \models \varphi_j$ for all j. 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, $X_n = \{q \in Q \mid q_0 \xrightarrow{\sigma} q \text{ in } \mathcal{C}\}$.*

Proof. Given $\sigma = (a_1,t_1)\cdots(a_n,t_n)$ and $X_0 = \{q_0\}$, for $j \in \{1,\ldots,n\}$, we define inductively $T_j = T(X_{j-1},a_j)$, $T'_j = \{(q,\varphi,a_j,q') \in T_j \mid \sigma, j \models \varphi\}$ and $X_j = \mathrm{target}(T'_j)$. Observe that $X_{j-1} \xrightarrow{\varphi(T'_j,T_j),a_j} X_j$ is a transition of $\mathcal{C}^{univ}$. Also by definition of T'_j, for all $T' \subseteq T_j$, we have

$$\sigma, j \models \varphi(T',T_j) \text{ if and only if } T' = T'_j \tag{1}$$

Using the "if" part above, we obtain that $X_0 \xrightarrow{\varphi(T'_1,T_1),a_1} X_1 \cdots \xrightarrow{\varphi(T'_n,T_n),a_n} X_n$ is a run of $\mathcal{C}^{univ}$ on σ. Conversely, we show by induction that this is the unique run of $\mathcal{C}^{univ}$ on σ starting from $X_0 = \{q_0\}$. Let $X_0 \xrightarrow{\varphi_1,a_1} X'_1 \xrightarrow{\varphi_2,a_2} \cdots X'_n$ be any such run. Suppose $X'_{j-1} = X_{j-1}$. Then we show that $\varphi_j = \varphi(T'_j,T_j)$ and $X'_j = X_j$. By definition of a run, we have $X_{j-1} \xrightarrow{\varphi_j,a_j} X'_j$ and $\sigma, j \models \varphi_j$. But by the definition of a transition, there exists $T' \subseteq T(X_{j-1},a_j) = T_j$ such that $\varphi_j = \varphi(T',T_j)$ and $X'_j = \mathrm{target}(T')$. Thus, $\sigma, j \models \varphi(T',T_j)$ which by Equation (1) implies that $T' = T'_j$. Thus, we conclude $\varphi_j = \varphi(T'_j,T_j)$ and $X'_j = \mathrm{target}(T') = \mathrm{target}(T'_j) = X_j$.

For the second statement, we prove both inclusions. First, if $q_0 \xrightarrow{\sigma} q$ in $\mathcal{C}$, let $q_0 \xrightarrow{\varphi_1,a_1} \cdots \xrightarrow{\varphi_n,a_n} q_n = q$ with $\sigma, j \models \varphi_j$ for $1 \leq j \leq n$. Using the notations above we show that for all $j \in \{0,\ldots,n\}$, $q_j \in X_j$. Clearly $q_0 \in X_0$. Assume $q_{j-1} \in X_{j-1}$. Then we have $(q_{j-1},\varphi_j,a_j,q_j) \in T'_j$. Hence we conclude that $q_j \in X_j = \mathrm{target}(T'_j)$.

Conversely, for all j and for all $q_j \in X_j$ we show that $q_0 \xrightarrow{(a_1,t_1)\cdots(a_j,t_j)} q_j$ is a run of $\mathcal{C}$. The proof is by induction on j. $j = 0$ is obvious. Assume $j > 0$ and let $q_j \in X_j$. Then there exists $(q_{j-1},\varphi_j,a_j,q_j) \in T'_j$, i.e., $q_{j-1} \in X_{j-1}$ and $\sigma, j \models \varphi_j$. By the induction hypothesis we have $q_0 \xrightarrow{(a_1,t_1)\cdots(a_{j-1},t_{j-1})} q_{j-1}$. This implies that $q_0 \xrightarrow{(a_1,t_1)\cdots(a_{j-1},t_{j-1})} q_{j-1} \xrightarrow{a_j,t_j} q_j$ is a run of $\mathcal{C}$. □

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 1. *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 $\mathcal{B}^B_{\mathcal{C}}$ from an MSC-ECA $\mathcal{C} = (Q, Act, \delta, q_0, F)$ such that $\mathcal{B}^B_{\mathcal{C}}$ and $\mathcal{C}$ are equivalent when restricted to B-bounded channels.

Let $K = \max\{k \mid \mathrm{Y}^k_p \in I$ occurs in some guard in $\delta\}$. A state of $\mathcal{B}^B_{\mathcal{C}}$ is either a dead state denoted $\bot$ 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 $Y = \{y^i_p \mid p \in Proc, 0 \le i < K\}$ and $Z = \{z^i_{p,q} \mid p, q \in Proc, 0 \le i \le B\}$. We will reset clock y^i_p when executing the i^{th} p-event mod K. Also, $z^i_{p,q}$ 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 $\delta_{\mathcal{B}^B_{\mathcal{C}}}$ 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 $\mathcal{B}^B_{\mathcal{C}}$.

(Tr1) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\text{true},a,\emptyset} \bot$ is in $\mathcal{B}^B_{\mathcal{C}}$ 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 $\mathcal{B}^B_{\mathcal{C}}$ 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 $\mathrm{Y}^k_p \in I$ 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 $\mathrm{Y}^k_p \in I$ is replaced as above and $\mathrm{Msg}^{-1} \in I$ is replaced with $z_{q,p}^{\alpha'_{q,p}} \in I$.

(Tr3) $\bot \xrightarrow{\text{true},a,\emptyset} \bot$ is in $\mathcal{B}^B_{\mathcal{C}}$ 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 $\#_{q?p}(w) \le \#_{p!q}(w)$. This weak form does not require the sequence of received messages to be a prefix of the sequence of the sent messages—it only demands that at any point, the number of messages received does not exceed the number of messages sent. Let $\mathrm{TW}^{B,\mathrm{wf}}_{Act}$ denote the set of timed words $\sigma \in \mathrm{TW}_{Act}$ which are both wwf and B-bounded.

We can immediately observe some invariant properties that are maintained by the above transitions. Let $\mathfrak{s}_0 \xrightarrow{\varphi_1,a_1,R_1} \cdots \xrightarrow{\varphi_m,a_m,R_m} \mathfrak{s}_m$ for $m \geq 0$ be a path in $\mathcal{B}_\mathcal{C}^B$ from the initial state $\mathfrak{s}_0$ to some state $\mathfrak{s}_m \neq \bot$. Then, for $\mathfrak{s}_m = (s_m, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$,

1. $b_p = 1$ if $|\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in Act_p\}| \geq K$ and $b_p = 0$ otherwise.
2. $n_p = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in Act_p\}| \bmod K$
3. $\alpha_{p,q} = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in q?p\}| \bmod (B+1)$
4. $\beta_{p,q} = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in p!q\}| \bmod (B+1)$

On the other hand, suppose $\mathfrak{s}_0 \xrightarrow{\varphi_1,a_1,R_1} \cdots \xrightarrow{\varphi_m,a_m,R_m} \mathfrak{s}_m$ for $m \geq 0$ is a path in $\mathcal{B}_\mathcal{C}^B$ from the initial state $\mathfrak{s}_0$ to $\mathfrak{s}_m = \bot$. Then, either σ is not wwf or it exceeds the bound B for some channel.

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 1. *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 $\mathrm{TW}_{Act}^{B,\mathrm{wf}}$.*

Further if $\mathcal{C}$ is finite so is $\mathcal{B}_\mathcal{C}^B$.

Proof. 1. 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, $\mathcal{L}_{tw}(\mathcal{C}) \cap \mathrm{TW}_{Act}^{B,\mathrm{wf}} \subseteq \mathcal{L}_{tw}(\mathcal{B}_\mathcal{C}^B)$ follows immediately from the following claim.

Claim. 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 $\mathcal{B}_\mathcal{C}^B$

We define inductively the valuation sequence for the run through π': ν_0 is the valuation mapping all clocks to 0, and $\nu_i = (\nu_{i-1} + t_i - t_{i-1})[R_i \to 0]$ for $1 \leq i \leq m$. To establish Claim 3.2 we show for all $i \in \{1, \ldots, m\}$ that $(\nu_{i-1} + t_i - t_{i-1}) \models \varphi'_i$.

There are three cases:

1. φ'_i contains $z_{p,q}^k \in I$ where $k = \alpha'_{p,q} = |\{a_\ell \mid 1 \leq \ell \leq i \wedge a_\ell \in q?p\}| \bmod (B+1)$. From the definition of the transition, φ_i must contain $\mathrm{Msg}^{-1} \in I$. Since, $\sigma, i \models \varphi_i$ we have $t_i - t_j \in I$, where j is the index of the matching send: $a_j = p!q(m), a_i = q?p(m), 1 \leq j \leq i$ and $|\{a_\ell \mid 1 \leq \ell \leq j \wedge a_\ell \in p!q\}| = |\{a_\ell \mid 1 \leq \ell \leq i \wedge a_\ell \in q?p\}|$. Thus, $k = |\{a_\ell \mid 1 \leq \ell \leq j \wedge a_\ell \in p!q\}| \bmod (B+1)$. Using the invariant at state $\mathfrak{s}_j$, we get $z_{p,q}^k \in R_j$. Using the invariant at $\mathfrak{s}_i$, we can replace $\mathrm{Msg}^{-1} \in I$ by $z_{p,q}^k \in I$ in φ'_i. Moreover, $z_{p,q}^k \notin R_\ell$ for $j < \ell \leq i$—otherwise, the number of events labelled $p!q$ between j and ℓ would be B more than the number of events labelled $q?p$ between j and i (and therefore ℓ). This implies that the channel was full and so, at ℓ, transition *(Tr1)* is enabled which means that transition *(Tr2)* cannot be fired, which contradicts the transition at i. Now, $z_{p,q}^k \in R_j$ implies $\nu_j(z_{p,q}^k) = 0$, and $z_{p,q}^k \notin R_\ell$ for $j < \ell \leq i$ implies that $(\nu_{i-1} + t_i - t_{i-1})(z_{p,q}^k) = \nu_j(z_{p,q}^k) + t_i - t_j = t_i - t_j$. So, we have $(\nu_{i-1} + t_i - t_{i-1}) \models (z_{p,q}^k \in I)$.

2. We will show that φ' cannot contain false. If φ' contains false, then $b_p = 0$ and there exists $\mathrm{Y}_p^k \in I$ in φ such that $k > n_p$. But $b_p = 0$ implies that K events have not been seen and so we are trying to relate two events that are k apart when we have not seen k events on p. This contradicts the fact that $\sigma, i \models \mathrm{Y}_p^k \in I$, so this cannot happen.
3. φ'_i contains $y_p^\ell \in I$, Then $\ell = (K + n'_p - k) \bmod K$ and $\mathrm{Y}_p^k \in I$ is in φ_i. Consider the event j such that the number of p-events between j and i is k. Such an event exists since either $n'_p > k$ or $b_p = 1$ (which means that $K > k$ many p-events have been seen). But if $k < n'_p$, then $\ell = n'_p - k$ and so the value of n_p-component at j is ℓ. If $k > n'_p$, then at j, we have $\ell = K + n'_p - k < K$. Thus in both cases, y_p^ℓ was reset at j and not reset again between j and i. Again, $\sigma, i \models \varphi_i$ implies that $t_i - t_j \in I$ and so $(\nu_{i-1} + t_i - t_{i-1})(y_p^\ell) = \nu_j(y_p^\ell) + t_i - t_j = t_i - t_j \in I$. Thus, $(\nu_{i-1} + t_i - t_{i-1}) \models y_p^\ell \in I$.

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 $\mathcal{C}$. This follows by considering each case for the guards and observing that if $\nu_{i-1} + t_i - t_{i-1} \models \varphi'_i$ then $\sigma, i \models \varphi_i$ in each case.

2. We have already noted that if a timed word σ has a run through a path of $\mathcal{B}_\mathcal{C}^B$ reaching the dead state $\bot$ then σ is either not wwf or not B-bounded. Conversely, assume that σ is either not wwf or not B-bounded and let σ' be the greatest prefix of σ which is both wwf and B-bounded. Since $\mathcal{C}$ is complete, the timed word σ' has a run through a path π of $\mathcal{C}$. As above we deduce that σ' has a run through a corresponding path π' of $\mathcal{B}_\mathcal{C}^B$. The next letter of σ will violate either the B-bound or the wwf condition. Hence the run reaches $\bot$ with this next letter and loops on $\bot$ until the end of σ. □

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

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

Theorem 1. *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. It is split in three main steps that are described in the following sections.

4.1 TCMSG to an infinite MSC-ECA

In this section, from a TCMSG, we construct an MSC-ECA (with infinitely many states) which accepts exactly the same set of timed linearisations. We start with a definition and a remark. For an MSC $M = (E, \leq, \lambda)$ over Act, recall that a *cut* c of M over Act to be a subset of the events E which is closed under the partial order $\leq$. That is, $e \in c$, $e' \leq e$ implies that $e' \in c$. This can of course be lifted to MSCs generated by a path π, namely M^π. We also recall that any event of E^π is of the form $(e, \rho u)$ where $\rho u \preceq \pi$ and $e \in E^u$. Indeed, keeping the prefix of the path along with the event uniquely identifies the event's occurence in the path.

For a fixed TCMSG $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, \mathit{EdgeC})$, where $G = (V, \rightarrow, v_{in}, V_F)$, we define the infinite MSC-ECA denoted $\mathcal{C}_{\mathfrak{G}}$. A *global state* of $\mathcal{C}_{\mathfrak{G}}$ is a pair $\overline{s} = (\pi, C)$ where

- π is a path in G.
- $C \subseteq E^\pi$ is a cut of M^π

Now, an event (e, ρ) is said to have been *executed* in $\overline{s}$ if $(e, \rho) \in C$. The event is said to be *enabled* in $\overline{s}$ if it has not been executed, i.e., $(e, \rho) \notin C$, and all the events below it (in the partial order) have been executed, i.e., for all $(e', \rho') \in E^\pi$ with $(e', \rho') <^\pi (e, \rho)$, we have $(e', \rho') \in C$.

A state $\overline{s} = (\pi, C)$ is *initial* if π is any path in G from an initial state to a final state and C is empty. It is *final* if $C = E^\pi$. We denote the set of all states of this global system by $Q_{\mathfrak{G}}$.

Now, the transitions can be defined by saying that at any state we execute an enabled event. We have $\overline{s} = (\pi, C) \xrightarrow{\varphi, a} \overline{s}' = (\pi, C')$ if there exists an event $(e, \rho u)$ enabled in $\overline{s}$ such that $\lambda^u(e) = a$ and

- $C' = C \uplus \{(e, \rho u)\}$
- the guard φ checks all local and edge constraints that are matched here,

$$\varphi = \Bigg(\bigwedge_{e' \in E^u, I \in \mathcal{I} | \tau^u(e', e) = I} \varphi(u, e', e, I) \Bigg) \wedge \varphi^{edge} \text{ where,} \tag{2}$$

$$\varphi(u, e', e, I) = \begin{cases} \mathrm{Msg}^{-1} \in I & \text{if } \exists p, q, p \neq q \text{ s.t. } e' <^u_{qp} e \\ \mathrm{Y}^k_p \in I & \text{if } e, e' \in E^u_p, |\{e'' \in E^u_p \mid e' \leq^u_{pp} e'' <^u_{pp} e\}| = k \end{cases} \tag{3}$$

$$\text{and } \varphi^{edge} = \begin{cases} \mathrm{Y}^1_p \in I & \text{if } \rho = \rho' u', \text{ and for some } p \in \mathit{Proc}, \text{ we have} \\ & \mathit{EdgeC}((u', u), p) = I \text{ and } e = \mathit{min}(E^u_p) \\ \text{true} & \text{otherwise} \end{cases} \tag{4}$$

Note that, in the transition above, the event $(e, \rho u)$ which is enabled in $\overline{s}$ becomes an executed event of $\overline{s}'$. Thus we can say that the transition $\overline{s} \xrightarrow{\varphi, a} \overline{s}'$ *executes* the event $(e, \rho u)$.

As before, a global run of $\mathcal{C}_{\mathfrak{G}}$ on a timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ is a sequence of transitions $\overline{s}_0 \xrightarrow{\varphi_1, a_1} \cdots \xrightarrow{\varphi_n, a_n} \overline{s}_n$ such that for each $j \in \{1, \ldots, n\}$, $\sigma, j \models \varphi_j$. Again a run is accepting if it starts at an initial state and ends in a global final state. We say a timed word σ belongs to $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}})$, if there is a global accepting run on σ.

Lemma 2. *We have the following relation between the timed languages of* $\mathcal{C}_{\mathfrak{G}}$ *and* $\mathfrak{G}$*:* $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathfrak{G}) = \{\sigma \mid \sigma$ is a timed linearisation of some TMSC T over Act, such that T realises some $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})\}$.

Proof. $(\subseteq)$ Let $\sigma = (a_1, t_1) \cdots (a_n, t_n) \in \mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}})$. Then there exists an accepting run

$$\overline{s}_0 \xrightarrow{\varphi_1, a_1} \cdots \xrightarrow{\varphi_n, a_n} \overline{s}_n$$

where for each $i \in \{1, \ldots, n\}$, $\sigma, i \models \varphi_i$ and $\overline{s}_{i-1} = (\pi_{i-1}, C_{i-1}) \xrightarrow{\varphi_i, a_i} (\pi_i, C_i) = \overline{s}_i$ executes some enabled event (e_i, ρ_i).

First, $\pi_0 = \cdots = \pi_n = \pi$ (say). Then, as $\overline{s}_0 = (\pi, C_0)$ is an initial state of the global system, π is a path from the initial vertex v_{in} to some final one in G. Therefore $\mathfrak{M}^\pi = (M^\pi, \tau^\pi) \in \mathcal{L}_{TC}(\mathfrak{G})$. Now, for each $i \in \{1, \dots, n\}$, $C_i = C_{i-1} \uplus \{(e_i, \rho_i)\}$ is a cut of M^π. Moreover, $C_n = E^\pi$ since $\overline{s}_n$ is final. From this we get that $(e_1, \rho_1) \cdots (e_n, \rho_n)$ is a linearisation of M^π and $\lambda^\pi(e_i, \rho_i) = a_i$ for all $i \in \{1, \dots, n\}$. Now consider the TMSC $T = (M^\pi, t)$ where we define t by $t((e_i, \rho_i)) = t_i$. Thus, $(a_1, t_1) \cdots (a_n, t_n)$ is a timed linearisation of T since $i < j$ implies $t(e_i, \rho_i) = t_i \leq t_j = t(e_j, \rho_j)$.

We are done if we show that T realises $\mathfrak{M}^\pi$. That is, for all $((e_i, \rho_i), (e_j, \rho_j)) \in \mathrm{dom}(\tau^\pi)$, we want to show that $|t(e_j, \rho_j) - t(e_i, \rho_i)| = t_j - t_i \in \tau^\pi((e_i, \rho_i), (e_j, \rho_j))$. We have two cases to handle:

- If $\rho_i = \rho_j = \rho v$ (say) then $\tau^\pi((e_i, \rho v), (e_j, \rho v)) = \tau^v(e_i, e_j) = I$. Then, first $\varphi(v, e_i, e_j, I)$ is in φ_j. Indeed, if $\tau^v(e_i, e_j) = I$ then one of the two following cases hold:
  - Either $e_i, e_j \in E_p^v$ for some $p \in \mathit{Proc}$. Then, $|\{e_\ell \in E_p^v \mid e_i \leq_{pp}^v e_\ell <_{pp}^v e_j\}| = k$ for some $k \in \mathbb{N}_{>0}$. Thus $\varphi(v, e_i, e_j, I) = \mathrm{Y}_p^k \in I$. At state j, $\sigma, j \models \varphi_j$ implies $\sigma, j \models \varphi(v, e_i, e_j, I)$ which implies that $\sigma, j \models \mathrm{Y}_p^k \in I$. Now, $e_\ell \in E_p^v$ such that $e_i \leq_{pp}^v e_\ell <_{pp}^v e_j$ if and only if $i \leq \ell < j$ such that $a_\ell \in \mathit{Act}_p$. Thus, by Definition *(D1)*, we conclude that $t_j - t_i \in I$.
  - Or $e_i <_{qp}^v e_j$ for some $p, q \in \mathit{Proc}, p \neq q$. Then, $\varphi(v, e_i, e_j, I) = \mathrm{Msg}^{-1} \in I$. Again, we have $\sigma, j \models \mathrm{Msg}^{-1} \in I$. Now, $e_i <_{qp}^v e_j$ implies that $\lambda^\pi(e_j, \rho_j) = a_j = p?q(m)$ for some $m \in \mathcal{M}$ and $\lambda^\pi(e_i, \rho_i) = a_i = q!p(m)$ is its matching send. Thus, $|\{a_\ell \mid 1 \leq \ell \leq i, a_\ell \in q!p\}| = |\{a_\ell \mid 1 \leq \ell \leq j, a_\ell \in p?q\}|$. Now, by Definition *(D2)* we conclude that $t_j - t_i \in I$.
- Otherwise, $\rho_j = \rho_i v, \rho_i = \rho v'$ for some ρ, $e_i = \mathit{max}(E_p^{v'})$ and $e_j = \mathit{min}(E_p^v)$ for some $p \in \mathit{Proc}$, then $\tau^\pi((e_i, \rho_i), (e_j, \rho_j)) = \mathit{EdgeC}((v', v), p) = I$. Then at stage j, we have $\varphi_j^{\mathit{edge}} = (\mathrm{Y}_p^1 \in I)$. Indeed, $a_i = \lambda^{v'}(e_i)$ is the last p-action before $a_j = \lambda^v(e_j)$ in σ. Thus, by Definition *(D1)*, $t_j - t_i \in I$ and so we are done.

($\supseteq$) Suppose $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, then there exists a path $\pi = v_1 \cdots v_m$ in G such that v_1 is an initial vertex and v_m is a final vertex and $\mathfrak{M} = \mathfrak{M}^\pi = (M^\pi, \tau^\pi)$. Now, suppose $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ is a timed linearisation of $T = (M^\pi, t)$ and T realises $\mathfrak{M}$. Then first we observe that $a_1 \cdots a_n \in \mathrm{lin}(M^\pi)$ and so there is $(e_1, \rho_1) \cdots (e_n, \rho_n)$ a linearisation of the events of M^π where for each i $\rho_i \preceq \pi$, $\lambda^\pi(e_i, \rho_i) = a_i$.

Then we can construct the run of the global system on this timed word. First, we define $C_i = \{(e_1, \rho_1) \cdots (e_i, \rho_i)\}$ and $\overline{s}_i = (\pi, C_i)$ for all $i \in \{0, \dots, n\}$, where $C_0 = \emptyset$. Then observe that (e_i, ρ_i) is enabled in $\overline{s}_{i-1}$. Thus there exists a transition $\overline{s}_{i-1} \xrightarrow{\varphi_i, a_i} \overline{s}_i$ that executes event (e_i, ρ_i). We show that $\sigma, i \models \varphi_i$ where φ_i is defined by the transition. Again there are two cases:

- Either φ_i contains an edge constraint, i,e., $\varphi^{\mathit{edge}} = (\mathrm{Y}_p^1 \in I)$ for some $p \in \mathit{Proc}$. In this case, by Condition 4, $\rho_i = \rho' v' v$, e_i is the first p-event in M^v and for some $j < i$, we have $\rho_j = \rho' v'$, e_j is the last p-event on $M^{v'}$ and $\mathit{EdgeC}((v', v), p) = I$. First, this implies that a_i is the next p-action with respect to a_j in σ. Also, $\tau^\pi((e_j, \rho_j), (e_i, \rho_i)) = I$ and since T realises $\mathfrak{M}$, we have $t(e_i, \rho_i) - t(e_j, \rho_j) \in I$ which implies that $t_i - t_j \in I$. By Definition *(D1)*, we conclude that $\sigma, i \models \varphi^{\mathit{edge}}$.
- Or there is a local constraint of a node of the form $\varphi(u, e_j, e_i, I)$, where $\rho_i = \rho u = \rho_j$ for some $j < i$ and $\tau^u(e_j, e_i) = I$. Again since T realises $\mathfrak{M}$, $t(e_i, \rho u) - t(e_j, \rho u) = t_i - t_j \in I$. By Condition 3, if the constraint is of the form $\mathrm{Msg}^{-1} \in I$, then $e_j <_{qp}^u e_i$ and otherwise

the constraint is of the form $\mathrm{Y}_p^k \in I$ where the number of p-events between e_j and e_i is k. Using Definition *(D2)* in the former case and Definition *(D1)* in the latter case, we conclude that $\sigma, i \models \varphi(u, e_j, e_i, I)$.

Thus, we have shown that $\sigma, i \models \varphi_i$ for all $i \in \{1, \ldots, n\}$ and therefore $\overline{s}_0 \xrightarrow{\varphi_1, a_1} \cdots \xrightarrow{\varphi_n, a_n} \overline{s}_n$ is a run of the global system on σ. Finally, the run ends in a global final state, since σ is a full linearisation of M^π. Thus, our proof is complete. □

4.2 Removing unexecuted nodes

We want to simulate the global run of a TCMSG in a finite way. So, instead of maintaining the whole path along the run, we want to maintain only the relevant portions, i.e., the nodes on which there is at least an event that has occurred.

For segments of nodes in the path that have not seen any event yet, we replace them by a special gap symbol #. Thus, having a # symbol between two nodes denotes that some (nonempty) sequence of nodes must be inserted here later.

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

This construction is formalized next. However, note that this construction is still infinite since we might still have unboundedly many completed nodes, i.e., nodes in which all events have been seen. In the next section, we describe how to perform a sequence of reductions to throw away such completed nodes from the current path. However, we have to be careful that the two conditions, in the infinite case above, are still maintained.

We start by observing that the cut C that we keep in a state in the simulation in the previous section is global. Thus, if we want to remove some nodes we would need to maintain the cut C locally within each node. To do this we break up each state $(u_1 \cdots u_n, C)$ into $(u_1, c_1) \cdots (u_n, c_n)$. Formally, we define the map Φ which we call *stratification* as follows:

$$\Phi((u_1 \cdots u_n, C)) = (u_1, c_1) \cdots (u_n, c_n)$$

where each $c_i \subseteq E^{u_i}$ is defined by $c_i = \{e \in E^{u_i} \mid (e, u_1 \cdots u_i) \in C\}$. Notice that each c_i is a cut of E^{u_i}. Φ is in fact a bijection since we also have the inverse map given by $C = \{(e, u_1 \cdots u_i) \in E^{u_1 \cdots u_n} \mid e \in c_i\}$.

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. As before, c 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* and we let $\Gamma = \mathit{ExtNodes} \uplus \{\#\}$.

Now, we construct our new automaton $\mathcal{C}^{\#}_{\mathfrak{G}}$. A state α of $\mathcal{C}^{\#}_{\mathfrak{G}}$ is an element of Γ^*. The initial state is $\alpha_0 = (\triangleright, \emptyset)\#(\triangleleft, \emptyset)$. Now, we lift the notion of events to *extended events* of a state in this new automaton. An *extended event* of $\alpha \in \Gamma^*$ is a pair $(e, \alpha_1(u, c))$ where $e \in E^u$ and $\alpha_1(u, c) \preceq \alpha$. 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) and 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 $e \in E_p^u$ then for all $\alpha_1'(u', c') \preceq \alpha_1$, we have $E_p^{u'} \subseteq c'$.

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 is *final* if it is a sequence of completed nodes.

We will need some notations to describe the set of processes that participate in node, path or a state. First, for a node $u \in V$, $\mathit{OProc}(u) = \{p \in \mathit{Proc} \mid E_p^u \neq \emptyset\}$ denotes the set of processes that participate (*occur*) in u. This is extended to V^* as a morphism. Also, with $\mathit{OProc}(u, c) = \mathit{OProc}(u)$ and $\mathit{OProc}(\#) = \emptyset$ it extends to Γ^*. In addition, for $\beta \in \Gamma^*$, $\mathit{EProc}(\beta)$ denoting the set of *executed* events in β, is given by the morphism defined by $\mathit{EProc}((u, c)) = \{p \in \mathit{Proc} \mid E_p^u \cap c \neq \emptyset\}$, $\mathit{EProc}(\#) = \emptyset$.

Now, the transitions can be defined by saying that at any state we can choose to execute an enabled event or add a new (extended) node to the state and then we must execute an enabled event on the new node. In fact, we always add a node by inserting it in a #.

Let us now define the *node insertion* operation which tells us how a node is inserted in a gap. Formally, this is defined 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.

Now, using this macro we can define the transition relation as follows. Formally, $\alpha \xrightarrow{\varphi, a} \alpha'$ is a transition in $\mathcal{C}_{\mathfrak{G}}^{\#}$ if there exists $\beta = \beta_1(u, c)\beta_2$ and an extended event $(e, \beta_1(u, c))$ enabled in β such that

- one of the two following conditions hold:
  (i) either $\beta = \beta_1(u, c)\beta_2 = \alpha$, 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 below conditions hold:
(T1) $a = \lambda^u(e)$
(T2) the guard φ must check all local and edge constraints, i.e.,

$$\varphi = \Bigg(\bigwedge_{e' \in E^u, I \in \mathcal{I} \mid \tau^u(e', e) = I} \varphi(u, e', e, I) \Bigg) \wedge \varphi^{\mathit{edge}} \text{ where,} \tag{5}$$

$$\varphi(u, e', e, I) = \begin{cases} \mathrm{Msg}^{-1} \in I & \text{if } \exists p, q, p \neq q \text{ s.t. } e' <_{qp}^u e \\ \mathrm{Y}_p^k \in I & \text{if } e, e' \in E_p^u, |\{e'' \in E_p^u \mid e' \leq_{pp}^u e'' <_{pp}^u e\}| = k \end{cases} \tag{6}$$

$$\text{and } \varphi^{\mathit{edge}} = \begin{cases} \mathrm{Y}_p^1 \in I & \text{if } \beta_1 = \beta_1'(u', c'') \text{ and for some } p \in \mathit{Proc}, \text{ we have} \\ & \mathit{EdgeC}((u', u), p) = I \text{ and } e = \mathit{min}(E_p^u) \\ \text{true} & \text{otherwise} \end{cases} \tag{7}$$

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

Observe as in the case of the automaton $\mathcal{C}_{\mathfrak{G}}$, 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 reachable states of this system satisfy some nice properties. To capture this we define the notion of a *valid state* of $\mathcal{C}^{\#}_{\mathfrak{G}}$.

A state α of $\mathcal{C}^{\#}_{\mathfrak{G}}$ is said to be *valid* if

(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 $u \rightarrow u'$ in G.
(V3) Executed events in α are *downward closed*. By this we mean that the following two conditions are satisfied:
  (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 $e \in E^u_p, e' \in E^{u'}_p$ for some $p \in \mathit{Proc}$, then $e' \in c' \implies e \in c$.

Proposition 2. *Every state of $\mathcal{C}^{\#}_{\mathfrak{G}}$ reachable from the initial state is valid.*

Proof. First note that the initial state is valid. Now, suppose α is valid and $\alpha \xrightarrow{\varphi,a} \alpha'$ we want to show that α' is valid as well. The first two properties follows from the node-insertion definition. The third follows from the definition of an enabled event. □

We may note however that the converse is not true in general, i.e., a valid state need not always be reachable.

Lemma 3. $\mathcal{L}_{tw}(\mathcal{C}^{\#}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}})$

Proof. We consider a morphism $\Psi : \mathit{ExtNodes}^* \rightarrow \Gamma^*$ defined by $(u, \emptyset) \mapsto \#$ and $(u, c) \mapsto (u, c)$ if $c \neq \emptyset$. We also define a reduction operation which acts on Γ^* and reduces consecutive multiple occurences of # into a single #. Formally, it is a rewrite operation where the rule is $\alpha_1\#\#\alpha_2 \xrightarrow{redn_{\#}} \alpha_1\#\alpha_2$. Then, for a state $\alpha \in \Gamma^*$, we denote by $\mathit{Red}_{\#}(\alpha)$ the state that we reach by a maximal sequence of repeated applications of this rule. We denote by Υ, the function that, given a global state $\overline{s}$ of $Q_{\mathfrak{G}}$, assigns the state of $\mathcal{C}^{\#}_{\mathfrak{G}}$ obtained as $\beta = (\triangleright, \emptyset)\mathit{Red}_{\#}(\Psi(\Phi(\overline{s})))(\triangleleft, \emptyset)$ where Φ is the stratification function defined earlier.

Now using the above definitions, we can relate accepting paths of the global semantics and accepting paths of the automaton $\mathcal{C}^{\#}_{\mathfrak{G}}$. The equality of the languages $\mathcal{L}_{tw}(\mathcal{C}^{\#}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}})$ follows immediately.

($\Longleftarrow$) Consider any global path of $\mathfrak{G}$, i.e.,

$$\overline{s}_0 \xrightarrow{\varphi_1,a_1} \overline{s}_1 \cdots \overline{s}_{n-1} \xrightarrow{\varphi_n,a_n} \overline{s}_n$$

where each $\overline{s}_i = (\pi, C_i)$. For all $i \in \{0, \ldots, n\}$, let $\beta_i = \Upsilon(\overline{s}_i)$. We will show that

$$\beta_0 \xrightarrow{\varphi_1,a_1} \beta_1 \cdots \beta_{n-1} \xrightarrow{\varphi_n,a_n} \beta_n$$

is a path of $\mathcal{C}^{\#}_{\mathfrak{G}}$.

Since $\overline{s}_0 = (\pi, C_0)$ is initial, we have $C_0 = \emptyset$ which implies that $\beta_0 = (\triangleright, \emptyset)\#(\triangleleft, \emptyset)$ which is the initial state of $\mathcal{C}^{\#}_{\mathfrak{G}}$. Fix $1 \le i \le n$ and let $\Phi(\overline{s}_{i-1}) = (u_1, c_1)\ldots(u_m, c_m)$ where $\pi = u_1 \cdots u_m$. Now, the transition $\overline{s}_{i-1} \xrightarrow{\varphi_i, a_i} \overline{s}_i$ executes some event $(e, u_1 \cdots u_j)$ which is enabled in $\overline{s}_{i-1}$. Then, $\overline{s}_i = (\pi, C_i)$ with $C_i = C_{i-1} \uplus \{(e, u_1 \cdots u_j)\}$. There are two cases to consider:

- Either $c_j \neq \emptyset$. In this case, we observe that $\beta_{i-1} = \Upsilon(\overline{s}_{i-1}) = \alpha_1(u_j, c_j)\alpha_2$ where we can write
$$\begin{aligned}\alpha_1 &= (\triangleright, \emptyset)\mathit{Red}_{\#}(\Psi((u_1, c_1)\cdots(u_{j-1}, c_{j-1})))\\ \alpha_2 &= \mathit{Red}_{\#}(\Psi((u_{j+1}, c_{j+1})\cdots(u_m, c_m)))(\triangleleft, \emptyset)\ .\end{aligned}$$
Then, we observe that $(e, u_1 \cdots u_j)$ is enabled in $\overline{s}_{i-1}$ implies that $(e, \alpha_1(u_j, c_j))$ is enabled in β_{i-1}. Thus, there exists a transition of $\mathcal{C}^{\#}_{\mathfrak{G}}$ which executes this event, namely $\beta_{i-1} \xrightarrow{\varphi'_i, a_i} \alpha_1(u_j, c'_j)\alpha_2$ where $c'_j = c_j \uplus \{e\}$. From Conditions (2–4) and (5–7), we deduce that $\varphi' = \varphi$. Then, $\Phi(\overline{s}_i) = (u_1, c_1)\cdots(u_j, c'_j)\cdots(u_m, c_m)$ by definition of C_i and so $\Upsilon(\overline{s}_i) = \alpha_1(u_j, c'_j)\alpha_2 = \beta_i$.
- Or $c_j = \emptyset$. That is, the event being executed is on a node that is not present in β_{i-1}. Then, there was a gap in β_{i-1} instead and we can write $\beta_{i-1} = \alpha_1\#\alpha_2$ where $\alpha_1\# = (\triangleright, \emptyset)\mathit{Red}_{\#}(\Psi((u_1, c_1)\cdots(u_j, c_j)))$. Now if $c_{j-1} = \emptyset$, then we let $\beta' = \alpha_1\#$ and else $\beta' = \alpha_1$. Similarly if $c_{j+1} = \emptyset$, then we let $\beta'' = \#\alpha_2$ and $\beta'' = \alpha_2$ otherwise. Then we can observe that $(e, \beta'(u_j, \emptyset))$ is enabled in $\beta'(u_j, \emptyset)\beta''$. Also, we have $\alpha_1\#\alpha_2 \xrightarrow{u_j} \beta'(u_j, \emptyset)\beta''$ since Conditions (I1), (I2) and (I3) hold. Indeed the latter two conditions follow from above, and if $\beta' = \alpha_1$ or $\beta'' = \alpha_2$, the presence of the edge in (I2), (I3) follows from the fact that the corresponding nodes are consecutive in π which is a path through G. Also if Condition (I1) is violated this would contradict the downward-closed property of the cut C_i.

  Thus there exists a transition in $\mathcal{C}^{\#}_{\mathfrak{G}}$, $\beta_{i-1} \xrightarrow{\varphi'_i, a_i} \beta_i = \beta'(u_j, c'_j)\beta''$ where $c'_j = \{e\}$. As above, we can conclude that $\varphi' = \varphi$. Now, $\Phi(\overline{s}_i) = (u_1, c_1)\cdots(u_j, c'_j)\cdots(u_m, c_m)$ where $c'_j \neq \emptyset$ and so $\Upsilon(\overline{s}_i) = \beta'(u_j, c'_j)\beta'' = \beta_i$.

Finally, since $\overline{s}_n$ is a final state of $\mathcal{C}_{\mathfrak{G}}$, $\beta_n = \Upsilon(\overline{s}_n)$ is a final state as well as it is a sequence of completed nodes. This completes the proof in one direction.

$(\Longrightarrow)$ For the converse consider an accepting path in $\mathcal{C}^{\#}_{\mathfrak{G}}$,
$$\alpha_0 \xrightarrow{\varphi_1, a_1} \alpha_1 \cdots \alpha_{n-1} \xrightarrow{\varphi_n, a_n} \alpha_n$$
where each $\alpha_i \in \Gamma^*$.

Then, α_n is final if it is a sequence of completed nodes, which we write as $(\triangleright, \emptyset)(u_1, c_1)\cdots(u_m, c_m)(\triangleleft, \emptyset)$. Then we claim that $\pi = u_1 \cdots u_m$ is a path in G from an initial state to a final state. This follows since this state is reachable and therefore valid and so Property (V2) holds (and from the definition of $\triangleright, \triangleleft$). Then, we will construct the global run inductively maintaining the invariant $\Upsilon(\overline{s}_i) = \alpha_i$ for all $i \in \{0, \ldots, n\}$.

At $i = 0$, $\overline{s}_0 = (\pi, C_0) = (\pi, \emptyset)$ and $\Upsilon(\overline{s}_0) = (\triangleright, \emptyset)\#(\triangleleft, \emptyset) = \alpha_0$. Suppose we have defined till $\overline{s}_{i-1} = (\pi, C_{i-1})$ such that $\Upsilon(\overline{s}_{i-1}) = \alpha_{i-1}$, with $\Phi(\overline{s}_{i-1}) = (u_1, c_1)\ldots(u_m, c_m)$. Consider $\alpha_{i-1} \xrightarrow{\varphi_i, a_i} \alpha_i$. Then again we have two cases:

- either the transition executes the event $(e, \beta_1'(u_j, c_j))$ which is enabled in $\alpha_{i-1} = \beta_1'(u_j, c_j)\beta_2' = \beta'$ where we let $\beta_1' = (\triangleright, \emptyset)\mathit{Red}_{\#}(\Psi((u_1, c_1)\cdots(u_{j-1}, c_{j-1})))$ and $\beta_2' = \mathit{Red}_{\#}(\Psi((u_{j+1}, c_{j+1})\cdots(u_m, c_m)))(\triangleleft, \emptyset)$.
- Or the transition inserts a node and then executes an enabled event, i.e., $\alpha_{i-1} = \beta_1\#\beta_2$ and $\beta_1\#\beta_2 \xrightarrow{u} \beta_1'(u, \emptyset)\beta_2' = \beta'$ and $(e, \beta_1'(u, \emptyset))$ is enabled in β'. Then $\beta_1' \in \{\beta_1, \beta_1\#\}$ and $\beta_2' \in \{\beta_2, \#\beta_2\}$. In π consider the first occurence of u, say u_j, which has no executed event in $\overline{s}_{i-1}$, i.e., $C_{i-1} \cap (E^{u_1\cdots u_j} \setminus E^{u_1\cdots u_{j-1}}) = \emptyset$. Thus, in this case, $c_j = \emptyset$.

Now, in both of the above cases, we claim that $(e, u_1\cdots u_j)$ is enabled in $\overline{s}_{i-1}$. Suppose not, choose a maximal event $(e', u_1\cdots u_{j'})$ which was not executed in $\overline{s}_{i-1}$, such that $(e', u_1\cdots u_{j'}) <^{\pi} (e, u_1\cdots u_j)$. This implies $j' \leq j$ and in fact, we have $j' < j$ since otherwise $e' <^{u_j} e$ which contradicts enabledness of $(e, \beta_1'(u_j, c_j))$ in β'. Thus, e' belongs to the same process as e. But then, there can't be any executed event in node $u_{j'}$, since if there was, the node would occur in α_{i-1} and so would contradict the fact that $(e, \beta_1'(u_j, c_j))$ is enabled in β' by violating Condition (E3). Now, if there was no executed event it would have been replaced by $\#$ in α_{i-1}. But then since we are simulating an accepting run of $\mathcal{C}_{\mathfrak{G}}^{\#}$, at some later transition, node $u_{j'}$ will be inserted in this $\#$. At that stage, we would violate Condition (I1) for node insertion since the process has seen an event, namely e to the right. Thus, we have a contradiction.

Once again, the existence of the enabled event immediately implies that there exists a transition that executes it in $\mathcal{C}_{\mathfrak{G}}$, namely $\overline{s}_{i-1} \xrightarrow{\varphi_i, a_i} \overline{s}_i$ such that $C_i = C_{i-1} \uplus \{(e, u_1\cdots u_j)\}$. Then we can also observe that $\Phi(\overline{s}_i) = (u_1, c_1)\cdots(u_j, c_j')$
$\cdots(u_m, c_m)$ and $c_j' = c_j \uplus \{e\}$. Thus, we conclude that $\Upsilon(\overline{s}_i) = \alpha_i$. □

In fact, we can strengthen the above lemma slightly without much change in the proof. If we restrict the above automaton to states that are both reachable and co-reachable even then the result holds. It turns out that this property of co-reachability is easy to capture in the automaton. 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 $\mathit{OProc}(\beta) \cap \mathit{EProc}((v, c')\alpha_2) = \emptyset$.

Corollary 2. *Consider the timed automaton obtained from $\mathcal{C}_{\mathfrak{G}}^{\#}$ by restricting to valid and completable states. Then, the timed language of this automaton is $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{\#})$.*

4.3 Removing completed nodes

As we mentioned earlier, from a state α we would like to obtain a finite abstraction of α, such that

1. the set of events left to be done are the same,
2. if $\alpha = \alpha_1\#\alpha_2$ where $\alpha_2 \in \Gamma^*$, then we want to preserve the information about the processes in $\mathit{EProc}(\alpha_2)$ so that if some nodes in α_2 are deleted we still know which processes must not be inserted in this gap.
   We accomplish this by enlarging the alphabet of nodes and $\#$ symbol with subsets of processes $P \subseteq \mathit{Proc}$. The idea is that this set P keeps track of the processes that are not allowed to participate in a node inserted on the left.
3. we preserve (do not throw away) the nodes around a $\#$ occurence in α and also nodes that start an edge constraint which needs to be verified later.

Formally, the set of states of our new automaton $\mathcal{C}_{\mathfrak{G}}^{fin}$ will be a finite subset of Π^* where $\Pi = \Gamma \cup 2^{Proc}$. Then, in our definition of the morphisms earlier we need to add $OProc(P) = P$, $EProc(P) = P$. Now, we define the reduction as a rewrite operation $\alpha \xrightarrow{redn} \alpha'$. There are two rewrite rules:

(R1) The first says that if two process sets are together they can be merged, i.e., $\alpha_1 P P' \alpha_2 \xrightarrow{redn} \alpha_1 (P \cup P') \alpha_2$.

(R2) Now, we define the rule that removes a completed extended-node (v, c) and replaces it by the set of processes participating in v, i.e., we have $\alpha_1 (v, c) \alpha_2 \xrightarrow{redn} \alpha_1 OProc(v) \alpha_2$ 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) $c = E^v$, i.e., all events in the node have been completed,

(C2.3) and one of the two following cases hold:

(i) either $\alpha_2 \in (v', c')\Pi^*$ and then for each $p \in Proc$ we must have either $E_p^v = \emptyset$ or $E_p^{v'} = \emptyset$ or $(c' \cap E_p^{v'}) \neq \emptyset$. In other words, if the first symbol of α_2 is an extended node (v', c') and there is an event in both E_p^v and $E_p^{v'}$, then some event in $E_p^{v'}$ has occured and so, the edge constraint has indeed been checked,

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

Remark 1. We can observe that, in some sense, the negation of Rule (R2) is an invariant of the reduction operation. More precisely, let $\alpha = \alpha_1 (u, c) \alpha_2$ be such that we cannot apply Rule (R2) to remove node (u, c) (given by its occurence $\alpha_1(u, c) \preceq \alpha$) and suppose $\alpha \xrightarrow{redn} \alpha'$. This, of course, implies that (u, c) (or rather, this occurence of (u, c)) is present in α' as well. Then, we can easily check that we cannot apply Rule (R2) to remove this node in α' either.

Lemma 4. *The rewrite system defined by the operation* $\xrightarrow{redn}$ *is confluent.*

Proof. Indeed it is easy to see that if the reduction rules apply on non-adjacent segments in a path, then they can be executed in any order. For instance, for $\beta \neq \varepsilon$, if we have $\alpha(u, c)\beta P P' \gamma \xrightarrow{redn} \alpha P'' \beta P P' \gamma$ where $P'' = OProc(u)$ and $\alpha(u, c)\beta P P' \gamma \xrightarrow{redn} \alpha(u, c)\beta(P \cup P')\gamma$, then of course $\alpha P'' \beta P P' \xrightarrow{redn} \alpha P'' \beta (P \cup P')\gamma$ and $\alpha(u, c)\beta(P \cup P') \xrightarrow{redn} \alpha P'' \beta (P \cup P')\gamma$. The interesting case is when two reduction rules apply on adjacent segments. Again, we may consider several subcases. If one of the reductions is by applying Rule (R1), then it is easy to handle since, in some sense, this rule does not depend on the context (i.e., the surrounding nodes/symbols). We now explicitly illustrate the subcase when we have two applications of Rule (R2) on adjacent nodes, i.e., let

– $\alpha(u, c)(u', c')\beta \xrightarrow{redn} \alpha(u, c)P'\beta$ where $P' = OProc(u')$ and
– $\alpha(u, c)(u', c')\beta \xrightarrow{redn} \alpha P(u', c')\beta$ where $P = OProc(u)$.

Then, from the first reduction we get $c' = E^{u'}$, $\varepsilon \neq \beta \notin \#\Pi^*$ and Condition (C2.3) holds with $\alpha_2 = \beta$. Using these and observing that $\alpha P \notin \Pi^* \#$, we can conclude that the first reduction is applicable after the second, i.e., $\alpha P(u', c')\beta \xrightarrow{redn} \alpha P P' \beta$. From the second reduction we have $c = E^u$ and $\varepsilon \neq \alpha \notin \Pi^* \#$. Now from these and the fact that Condition (C2.3)(ii) holds, we can conclude that the second reduction is applicable after the first, i.e., $\alpha(u, c)P'\beta \xrightarrow{redn} \alpha P P' \beta$. □

Using the above lemma we can conclude that, from any state α after any maximal sequence of reductions, we reach the same state which we denote by $\mathit{Red}(\alpha)$. Note that if $\alpha \xrightarrow{redn} \alpha'$, then $\mathit{EProc}(\alpha) = \mathit{EProc}(\alpha')$ and therefore, $\mathit{EProc}(\alpha) = \mathit{EProc}(\mathit{Red}(\alpha))$. In fact, from confluence, we derive some useful properties of the reduction operation,

(P1) $\mathit{Red}(\alpha_1 \# \alpha_2) = \mathit{Red}(\alpha_1) \# \mathit{Red}(\alpha_2)$.

(P2) $\mathit{Red}(\alpha_1\alpha_2) = \mathit{Red}(\mathit{Red}(\alpha_1)\alpha_2) = \mathit{Red}(\alpha_1 \mathit{Red}(\alpha_2)) = \mathit{Red}(\mathit{Red}(\alpha_1)\mathit{Red}(\alpha_2))$

(P3) Let $\alpha = \alpha_1(u,c)\alpha_2$ be such that this (u,c) (given by its occurence $\alpha_1(u,c)$) cannot be reduced in α, i.e., Rule (R2) cannot be applied. Then $\mathit{Red}(\alpha) = \gamma_1(u,c)\gamma_2$ where $\gamma_1(u,c) = \mathit{Red}(\alpha_1(u,c))$ and $(u,c)\gamma_2 = \mathit{Red}((u,c)\alpha_2)$.

Proof. The first two properties are self-evident. For the third, using Remark 1 we deduce that (u,c) is not deleted during the reductions. Now, let $\gamma_1(u,c) = \mathit{Red}(\alpha_1(u,c))$ and $(u,c)\gamma_2 = \mathit{Red}((u,c)\alpha_2)$. Then applying Property (P2) twice on α, we obtain $\mathit{Red}(\alpha_1(u,c)\alpha_2) = \mathit{Red}(\mathit{Red}(\alpha_1(u,c))\alpha_2) = \mathit{Red}(\gamma_1(u,c)\alpha_2) = \mathit{Red}(\gamma_1 \mathit{Red}((u,c)\alpha_2)) = \mathit{Red}(\gamma_1(u,c)\gamma_2)$. Now since $\gamma_1(u,c)$ and $(u,c)\gamma_2$ are already in reduced form and (u,c) cannot be deleted in $\mathit{Red}(\alpha)$, we obtain $\mathit{Red}(\gamma_1(u,c)\gamma_2) = \gamma_1(u,c)\gamma_2$. □

The set of *final* states of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ are states of the form $(\triangleright, \emptyset)P(\triangleleft, \emptyset)$ where $P \subseteq \mathit{Proc}$.

In the definition of a transition of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ we replace the final condition (T3) with the following condition:

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

Now, if we maintain the rest of the definition of a transition of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ to be the same as a transition of $\mathcal{C}^{\#}_{\mathfrak{G}}$, we can prove that $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ is a finite MSC-ECA which accepts the same timed language as $\mathcal{C}^{\#}_{\mathfrak{G}}$. We can also observe that in all reachable states of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$, Properties (V1), (V2) and (V3) continue to hold.

Lemma 5. *If $\mathfrak{G}$ is locally synchronized, then $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ as defined above is a finite MSC-ECA.*

Proof. We show that if $\mathfrak{G}$ is locally synchronized, then the number of states of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ is finite. For this, it is enough to show that the length of each reachable, completable state of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ is bounded. Note that by definition in every state in every extended node there is at least one executed event. We begin with some properties about a loop in a state which follow from the locally synchronized assumption.

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

Proof. Let $e \in c$ such that $e \in E^u_p$ for some $p \in \mathit{Proc}$. Since $\alpha(u,c)\beta(u,c')\gamma$ is completable, for each occurence of # in β, there exists $u_1 \cdots u_n \in V^*$ in G such that if we replace the # by $(u_1,\emptyset)\cdots(u_n,\emptyset)$, then we obtain a path β' such that $\alpha(u,c)\beta'(u,c')\gamma$ is a valid state.

Now, we can write $(u,c)\beta' = \beta_1(v,c'')\beta_2$ with $c'' \subsetneq E^v$. This follows, since either there is a # in β, and so for any node (v,c'') on the path inserted we have $c'' = \emptyset$, or else $\beta' = \beta$ and by assumption $(u,c)\beta$ is not completely executed. Now, let $e' \in (E^v \setminus c'')$ such that $e \in E^v_{p'}$ for some p'.

Consider the path $\widehat{\beta'}$ in G, obtained by restricting β' to its first component. Now, as $\mathfrak{G}$ is locally synchronized, in the communication graph of $M^{u\widehat{\beta'}}$ there exists a path from p' to

p. Then let this path be $p' = p_0 \to p_1 \to \cdots \to p_n = p$ for some $n \geq 1$. We call a process q *good* if there is an executed event and an unexecuted event on q in $(u,c)\beta'$. If q is good, then $q \in (\mathit{EProc}((u,c)\beta') \setminus \mathit{EProc}((u,c')\gamma))$. We will now show that there is some good process $q \in \{p_0, \ldots, p_n\}$.

Suppose, $p_n = p$ has an unexecuted event in $(u,c)\beta'$ then it is good and we are done. Otherwise, p must have completed its events in $(u,c)\beta'$ and so it must have received a message from p_{n-1}. Therefore, p_{n-1} has also taken part in $(u,c)\beta'$ since it must have sent the message that was received by p_n. Now if p_{n-1} has another event in $(u,c)\beta'$ which is unexecuted, then it is good and again we are done. Otherwise, we repeat this argument till we reach an executed event in $p_0 = p'$. But this implies that p' is good and so we are done. □

Claim. If $\alpha(u,c)\beta(u,c')\gamma$ is a valid state such that $(u,c)\beta(u,c')$ is completely executed and β has no $\#$, then $\alpha = \alpha'\#$.

Proof. Since $(u,c)\beta(u,c')$ is completely executed, the first occurence of node u, i.e., (u,c) would have been deleted unless $\alpha = \alpha'\#$ or $\beta = \#\beta'$. But since β does not contain $\#$ the latter case is not possible and so we are done. □

From the above claim we can conclude that after every two occurences of node u in a path, there must exist a $\#$ or the segment is not completely executed. Then, along with the previous claim this implies that we can bound the number of occurences of a node u in a path by $2|\mathit{Proc}|$. From which we can conclude that we have a bound of $(2|\mathit{Proc}|)|V|$ on the number of extended nodes in a path. But we know that each $\#$ or $P \subseteq \mathit{Proc}$ must have a node $u \in V$ next to it on the left, so we can conclude that the length of the path is $\mathcal{O}(|\mathit{Proc}||V|)$. Thus $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ is finite. □

Now, we will show that the timed language accepted by $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ is the same as the timed language accepted by $\mathcal{C}^{\#}_{\mathfrak{G}}$. We will accomplish this by defining a bisimulation between the states of the abstract automata $\mathcal{C}^{\#}_{\mathfrak{G}}$ and $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$. From this, we can conclude that their timed languages coincide. We define the relation $\rightsquigarrow$ between states of $\mathcal{C}^{\#}_{\mathfrak{G}}$ and $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$:

$$\alpha \rightsquigarrow \beta \text{ if } \beta = \mathit{Red}(\alpha) \tag{8}$$

Now, we have the lemma,

Lemma 6. *$\rightsquigarrow$ is a bisimulation on abstract automata $\mathcal{C}^{\#}_{\mathfrak{G}}$ and $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$*

Proof. Let α be a state of $\mathcal{C}^{\#}_{\mathfrak{G}}$ and β a state of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ such that $\alpha \rightsquigarrow \beta$, i.e., $\beta = \mathit{Red}(\alpha)$.
($\Longrightarrow$) In one direction we start from a move $\alpha \xrightarrow{\varphi,a} \alpha'$ in $\mathcal{C}^{\#}_{\mathfrak{G}}$ and show that there is a move $\beta \xrightarrow{\varphi,a} \beta'$ in $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$, where $\beta' = \mathit{Red}(\alpha')$. There are two broad cases to consider depending on whether the transition in $\mathcal{C}^{\#}_{\mathfrak{G}}$ extends the path or not.

- Suppose the path is extended. Then, we have $\alpha = \alpha_1\#\alpha_2 \xrightarrow{u} \alpha_1'(u,\emptyset)\alpha_2' = \alpha''$ where $\alpha_1' \in \{\alpha_1, \alpha_1\#\}$ and $\alpha_2' \in \{\alpha_2, \#\alpha_2\}$. Also, there exists an extended event $(e, \alpha_1'(u,\emptyset))$ enabled in α'' such that $\alpha' = \alpha_1'(u,c')\alpha_2'$ where $c' = \{e\}$. Then, we observe that
  1. we can write $\beta = \beta_1\#\beta_2$ where $\beta_1 = \mathit{Red}(\alpha_1)$ and $\beta_2 = \mathit{Red}(\alpha_2)$. This follows by Property (P1).

2. we have $\beta_1\#\beta_2 \xrightarrow{u} \beta'_1(u,\emptyset)\beta'_2 = \beta''$ where $\beta'_1 \in \{\beta_1, \beta_1\#\}$ and $\beta'_2 \in \{\beta_2, \#\beta_2\}$. Further $\beta'_1 = \beta_1$ if and only if $\alpha'_1 = \alpha_1$ and $\beta'_2 = \beta_2$ if and only if $\alpha'_2 = \alpha_2$. The existence of this node insertion move follows from the node insertion in $\mathcal{C}^{\#}_{\mathfrak{G}}$ above since we have $\mathit{OProc}(u) \cap \mathit{EProc}(\alpha_2) = \emptyset$, which implies that $\mathit{OProc}(u) \cap \mathit{EProc}(\beta_2) = \emptyset$ (since $\beta_2 = \mathit{Red}(\alpha_2)$). Notice that we also have for $i \in \{1,2\}$, $\beta'_i = \mathit{Red}(\alpha'_i)$ since $\beta_i = \mathit{Red}(\alpha_i)$.
3. $(e, \beta'_1(u,\emptyset))$ is enabled in β''. Indeed, Conditions (E1), (E2) hold since they hold for $(e, \alpha'_1(u,\emptyset))$. And if there exists $(\hat{e}, \hat{\beta}(\hat{u},\hat{c}))$ such that $e, \hat{e}$ are on the same process, $\hat{\beta}(\hat{u},\hat{c}) \preceq \beta'_1 = \mathit{Red}(\alpha'_1)$ and $\hat{e} \notin \hat{c}$, then $\hat{\beta}'(\hat{u},\hat{c}) \preceq \alpha'_1$ for some $\hat{\beta}'$. This contradicts the fact that $(e, \alpha'_1(u,\emptyset))$ is enabled in α''. Therefore Condition (E3) holds as well.

Then by definition of a transition, we have $\beta \xrightarrow{\varphi',a} \beta' = \mathit{Red}(\beta'_1(u,\{e\})\beta'_2)$ which executes this enabled event in $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$.

Now, we show that the same guard is used, i.e., $\varphi' = \varphi$. For this, observe that $\varphi = \varphi^{\mathit{edge}}$ and $\varphi' = \varphi'^{\mathit{edge}}$ since there are no local-constraints. Now $\varphi^{\mathit{edge}} = (\mathrm{Y}^1_p \in I)$ for some $p \in \mathit{Proc}$ if and only if $e = \mathit{min}(E^u_p)$, $\alpha'_1 = \alpha_1 = \alpha''_1(u',c'')$, $\mathit{EdgeC}((u',u),p) = I$. But now, the node u' cannot be removed during the reduction of α since it is next to a $\#$, so we have $\beta'_1 = \beta_1 = \beta''_1(u',c'')$ which implies that we have the constraint $\varphi'^{\mathit{edge}} = (\mathrm{Y}^1_p \in I)$.

Finally, we will be done with this case if we show that $\mathit{Red}(\alpha') = \beta'$. We have $\beta' = \mathit{Red}(\beta'_1(u,c')\beta'_2) = \mathit{Red}(\mathit{Red}(\alpha'_1)(u,c')\mathit{Red}(\alpha'_2))$. But by Property (P2) this is equal to $\mathit{Red}(\alpha'_1(u,c')\alpha'_2) = \mathit{Red}(\alpha')$ and so we are done.

- Else, it was not extended then there exists an enabled event $(e, \alpha_1(u,c))$ in α which is executed in the transition $\alpha \xrightarrow{\varphi,a} \alpha'$, where $\alpha = \alpha_1(u,c)\alpha_2, \alpha' = \alpha_1(u,c')\alpha_2$ with $c' = c \uplus \{e\}$ and φ is defined by Equation (T2). Then (u,c) is not completely executed and so it cannot be reduced in α. Thus by Property (P3), $\beta = \mathit{Red}(\alpha) = \gamma_1(u,c)\gamma_2$, where $\gamma_1(u,c) = \mathit{Red}(\alpha_1(u,c))$ and $(u,c)\gamma_2 = \mathit{Red}((u,c)\alpha_2)$. Now, $(e, \gamma_1(u,c))$ is enabled in β, since $(e, \alpha_1(u,c))$ was enabled in α, and Conditions (E1),(E2) and Condition (E3) follow as in the previous case. That is, if there exists $(\hat{e}, \hat{\beta}(\hat{u},\hat{c}))$ such that $\hat{\beta}(\hat{u},\hat{c}) \preceq \gamma_1$, then $\hat{\beta}'(\hat{u},\hat{c}) \preceq \alpha_1$ for some $\hat{\beta}'$.

  Thus, there exists a transition $\beta \xrightarrow{\varphi',a} \beta'$ that executes $(e, \gamma_1(u,c))$ in $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$. Again we check that $\varphi' = \varphi$. This follows as in the previous case except that we also need to check local constraints in φ'. But as the guards are local to the node (u,c) which is not deleted in β, this follows directly from the definition.

  It remains to show that $\mathit{Red}(\alpha') = \beta'$. Since $\alpha' = \alpha_1(u,c')\alpha_2$ is such that $c \subsetneq c' \subseteq E^u$, we have $\mathit{Red}(\alpha') = \mathit{Red}(\alpha_1(u,c')\alpha_2) = \mathit{Red}(\gamma_1(u,c')\gamma_2) = \beta'$. This follows because, every reduction that can be performed on α can be performed on α' and so performing a maximal sequence of reductions on α' is equivalent to performing all the reductions on α and then again perhaps performing a few more if the resulting state is not fully reduced (due to events in $(c' \setminus c)$).

$(\Longleftarrow)$ For the other direction, the result follows by observing that the enabled event that gets executed in the infinite system $\mathcal{C}^{\#}_{\mathfrak{G}}$ is obtained from the corresponding event in the finite system $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$. More formally, we assume that $\beta \xrightarrow{\varphi,a} \beta'$ is a transition in $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ and show that there is a transition $\alpha \xrightarrow{\varphi,a} \alpha'$ in $\mathcal{C}^{\#}_{\mathfrak{G}}$.

Let the transition in $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$ execute the event $(e, \beta_1(u,c))$ enabled in $\beta = \beta_1(u,c)\beta_2$. Indeed there is another case where the executed event is not in β and so we need to perform a node insertion before we obtain the enabled event. But as this case follows by the same arguments (and is in fact simpler due to presence of $\#$), we only consider the first case.

Let $\alpha_1(u, c'')$ be the least prefix of α such that $e \notin c''$. Then (u, c'') is not removed by the reduction operation. Since $\beta = \mathit{Red}(\alpha)$ and $(e, \beta_1(u, c))$ is enabled in β, we deduce from (E3) that $c'' = c$ and $\mathit{Red}(\alpha_1(u, c)) = \beta_1(u, c)$. Now we claim that $(e, \alpha_1(u, c))$ is enabled in α. Conditions (E1),(E2) hold since they hold for $(e, \beta_1(u, c))$. Suppose Condition (E3) did not hold, then for $p \in \mathit{Proc}$ such that $e \in E^u_p$, there exists an event $(e', \hat{\alpha_1}(v, c'))$ with $e' \in (E^v_p \setminus c')$ and $\hat{\alpha_1}(v, c') \preceq \alpha_1$. Again, $\mathit{Red}(\hat{\alpha_1}(v, c')) = \hat{\beta_1}(v, c') \prec \beta_1$ (since (v, c') cannot be removed by reductions). But then $e' \in (E^v_p \setminus c')$ is a contradiction of Condition (E3) on $(e, \beta_1(u, c))$. Thus all the conditions hold and $(e, \alpha_1(u, c))$ is enabled in α.

Thus, we can conclude that there is a transition that executes $(e, \alpha_1(u, c))$ in $\mathcal{C}^{\#}_{\mathfrak{G}}$, i.e., $\alpha \xrightarrow{\varphi', a} \alpha'$. The fact that $\varphi' = \varphi$ and $\beta' = \mathit{Red}(\alpha')$ follows exactly as in the previous direction so we are done. □

Corollary 3. $\mathcal{L}_{tw}(\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathcal{C}^{\#}_{\mathfrak{G}})$

Proof. From the above bisimulation at the symbolic level of paths, we deduce easily that the timed language of $\mathcal{C}^{\#}_{\mathfrak{G}}$ is equal to the timed language of $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$. □

Proof. (of Theorem 1) Given a locally synchronized TCMSG $\mathfrak{G}$, consider the finite MSC-ECA $\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}$. Then, by using the above corollary, Lemma 3 and Lemma 2, we conclude that $\mathcal{L}_{tw}(\mathcal{C}^{\mathit{fin}}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathfrak{G})$. □

5 Solving the model checking problem

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

Theorem 2. *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 $\mathcal{L}_{tw}(\mathcal{A}) \cap (\mathrm{TW}_{\mathit{Act}} \setminus \mathcal{L}_{tw}(\mathfrak{G})) = \emptyset$. By Theorem 1 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 $\mathcal{C}' = \mathcal{C}^{\mathit{univ}}_2$ such that by Corollary 1 we have $\mathcal{L}_{tw}(\mathcal{C}') = \mathrm{TW}_{\mathit{Act}} \setminus \mathcal{L}_{tw}(\mathcal{C}) = \mathrm{TW}_{\mathit{Act}} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

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: $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}}$. 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 1. We get $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = (\mathrm{TW}_{\mathit{Act}} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}}) \cup (\mathcal{L}_{tw}(\mathcal{C}') \cap \mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}}) = (\mathrm{TW}_{\mathit{Act}} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}}) \cup (\mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}} \setminus \mathcal{L}_{tw}(\mathfrak{G}))$. Using $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{\mathit{Act}}$ we deduce $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = \mathrm{TW}_{\mathit{Act}} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

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. R. Alur and D. Dill: A Theory of Timed Automata. *Theor. Comput. Sci.*, **126** (1994) 183–225.
2. R. Alur, G. Holzmann and D. Peled: An analyzer for message sequence charts. *Software Concepts and Tools*, **17(2)** (1996) 70–77.

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.