Specification and Verification for Distributed and Timed Systems

Akshay Sundararaman

PhD Thesis
Defended on 02 July 2010

Abstract

Our goal is to use formal methods to reason about systems where time and concurrency play a significant role. We are interested in checking if the behaviours exhibited by an implementation conform to those stipulated by the specification in a timed and distributed system.

To describe the behaviours of distributed systems which operate on a global time, we introduce two notions of timed partial orders. The first, timed message sequence charts (TMSCs) are concrete models used to describe system executions. The second, time constrained message sequence charts (TCMSCs) are more abstract and represent families of TMSCs. For appropriate formalisms of implementation (timed message passing automata) and specification (monadic second order logic) over TMSCs, we obtain an expressive equivalence.

Infinite collections of TCMSCs can also be specified using time constrained message sequence graphs (TCMSGs). We address two problems that arise in this setting, consistency and coverage. Consistency asks if every run of the implementation is compatible with some TCMSC generated by the TCMSG. Coverage asks if every TCMSC generated by the TCMSG is witnessed by the implementation.

In the second part of the talk, we consider an alternate system model where clocks in different components of a distributed system evolve at different rates. We look at two natural semantics. The universal semantics captures behaviours that hold under any choice of clock rates, while the existential semantics captures those the system may exhibit under some choice of clock rates. We show that the existential semantics always exhibits a regular set of behaviours. However, for universal semantics, checking emptiness turns out to be undecidable. As an alternative, we propose a reactive semantics that lets us check positive specifications and yet describes a regular set of behaviours.

Acknowledgements

First of all, I would like to express my profound gratitude to my supervisors for all their guidance and support during my PhD. I have learnt much about the rigors of academic thinking and research from my advisor Paul Gastin. My co-advisor Madhavan Mukund has been a mentor and friend throughout. I am also very grateful to Benedikt Bollig and Narayan Kumar for their unfailing encouragement and for the various discussions we have had. I thank my thesis reviewers and jury members for their time and effort.

I thank the faculty and students at LSV, Chennai Mathematical Institute and the Institute of Mathematical Sciences, the administrative staff at these institutes and the good people at the International Students Relations office at Cachan for their invaluable help and support. Ramanujam, Kamal Lodaya and Meena Mahajan for all their generous advice and for helping me with several critical decisions along the way. And Hubert Comon-Lundh for ensuring that my Masters course lectures were in English, when I first joined ENS-Cachan!

I am indebted to all my amazing friends in Cachan, Paris, Chennai and Bangalore, for making these years, the most memorable ones of my life. I owe much to Alexis and Laurence (and co.) for the fact that I not only survived my French experience but thrived in it. Finally, I thank my family for being there, my last line of defense.

Contents

List of Figures

1 Introduction

1.1 Background and motivation

In today's world, we encounter computational devices all around us. These devices do not act in isolation but interact in increasingly complex ways. For example, Automatic Teller Machines (ATMs), online banking systems, car braking systems are all composed of several components that need to communicate with each other over an extended period of time. Even a simple system of a railway gate controller requires us to manage the interaction between the gate and the signal. Designing these systems and analyzing how they function (and evolve) is an important focus of computer science.

Two crucial aspects in this study are concurrency and timing. Concurrency plays an important role since systems usually consist of independent components that interact periodically to coordinate their behaviour. Timing considerations play an important role in describing how these interactions proceed.

Of course, there are many ways to model concurrency as well as time in systems. Concurrency is typically modelled by considering distributed system components that synchronize. This synchronization can be achieved by using common actions or by exchanging messages, for instance through fifo channels or through shared variables. To describe how the behaviour of the system depends on time, we use devices such as clocks and time-outs. Combining concurrency and time gives rise to a new set of questions. Does there exist a global clock that synchronizes all the distributed components? Do different distributed components need to refer to the same clocks or even the same time?

1.2 The starting point

Our goal is to use formal methods to reason about systems where time and concurrency play a significant role. The first challenge is to choose a suitable formalism that admits automated analysis. For instance, if a system exhibits regular, finite-state behaviour, we can use *model checking* to efficiently explore the state space

and determine various behavioural properties.

Model checking is the paradigm where systems are abstracted into models that are then automatically checked for correctness against formal specifications. This approach has been quite successful in handling industrial-sized verification problems. Finite-state automata provide an intuitively appealing machine model for generating regular behaviours. These regular behaviours can be represented as a set of words over an alphabet. Monadic second order logic is an elegant language to describe abstract properties of sets of words. The Büchi-Elgot Theorem [20,37] links the two formalisms: a behaviour can be described by a finite-state automaton if and only if it can be expressed in monadic second order logic. This correspondence is effective and forms the basis for model checking behavioural properties of finite-state systems. We would like to lift this approach to the timed and distributed setting.

However, while trying to describe the behaviours of a timed and distributed system, we find that words are not good enough abstractions. First of all, when we describe the behaviour of a distributed system as a word, we implictly sequentialize events that occur on different processes. Thus, when events occur independently (on different processes), the behaviour is represented by the set of all interleavings of the events. If we have n events, to know that they are independent, we will have to check $n!$ interleavings in this representation. This is sometimes referred to as the combinatorial or state explosion problem. Also, by using the interleaving semantics, we end up "flattening" the model into a large global system where concurrency is replaced by nondeterminism. Thus, we are motivated to look at partial orders which make precise the dependence between the actions and allow for independence of certain actions. There are many popular notions which formalize this including traces, event structures and message sequence charts.

Further, in a timed system, there is an infinite set of time points when an event might happen. This can be modelled by attaching to each event a real-valued variable denoting the time at which it occurred. Such an extension in the sequential setting is called a timed word. It is natural to extend this to the distributed setting by defining partial orders with time stamps.

1.3 Models in the timed and distributed setting

As motivated above, we are interested in checking if the behaviours exhibited by an implementation conform to the behaviours stipulated by a specification in a timed and distributed system. Potentially, this allows us to specify and verify properties of the system in an automated framework.

The crux of this approach lies in extending the notion of behaviours to timed partial orders. Once this is accomplished, we can introduce appropriate abstract formalisms for implementation and specification and perform such an analysis.

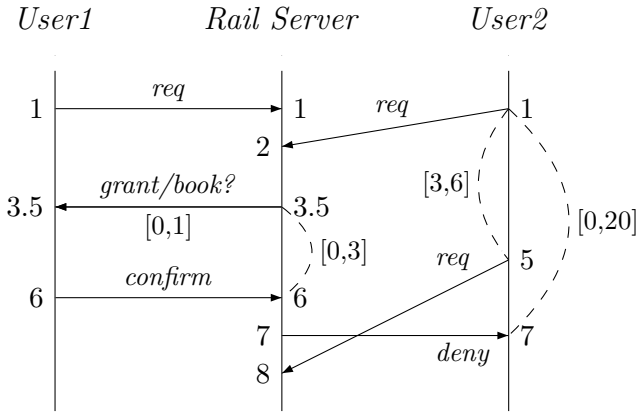


Figure 1.1: A message sequence chart with timing information

1.3.1 Models for behaviours

We use message sequence charts as our basic models of partially ordered executions. A message sequence chart (MSC) is a visual representation of the interactions between various components or processes in a distributed system. The processes are drawn as vertical lines and interpreted as time axes on which send or receive events take place. The components interact through messages which are depicted as arrows from send to receive events on the process lines.

MSCs are widely used in industry and have been standardized in [44, 45]. They serve as documentation of design requirements that are referred throughout the design process including final system integration and acceptance testing.

Formally, MSCs can be described as partially ordered sets of events in which each event is labelled as a send or a receive action. We introduce two notions of MSCs with timing information. We first consider timed MSCs which are just MSCs with time-stamps at events (as in timed words). These are ideal to describe real-time system executions, while keeping the causal relation between events explicit. Next, we consider MSCs with timing constraints or time-constrained MSCs, where we associate time-intervals to some pairs of events (instead of attaching time-stamps to individual events). The endpoints of the interval give us the upper and lower bounds on the time allowed to elapse between the events. This formalism is more suitable for specification and also useful for describing a possibly infinite family of timed MSCs in a finite way.

Example 1. Consider the scenario presented in Figure 1.1 which depicts part of an interaction between two users and a railway ticket-booking server. The vertical line orders the events on a single process while the arrows represent messages between the processes. Thus, in the scenario shown, two users *User1* and *User2* send booking requests to the *Server*. The *Server* grants *User1*'s request since it is received first. Then, *User1* confirms her booking which leads to the server

denying the booking to *User2*. Meanwhile, *User2* repeats his request which reaches the *Server*.

If we ignore the time stamps and intervals mentioned we have an MSC. Now, if we include the time stamps mentioned at each event, we obtain a timed MSC which tells us exactly when each event took place. For instance, we can see that the message *confirm* was sent by *User1* at time instant 6 and received instantaneously by the *Server*. If we consider only the time intervals mentioned between events, we obtain a time-constrained MSC. In the above scenario, for instance, we require that the time difference between two consecutive requests by *User2* to *Server* is atmost 6 time units and at least 3. We will revisit this example in Sections 3.1 and 3.3 in more detail.

1.3.2 Models for implementation

Next, we consider automaton models that implement the behaviours described above. As an implementation for MSCs, we consider message passing automata (MPA) or communicating finite-state machines introduced in [19], which are a fundamental model for concurrent systems and communicating protocols. An MPA consists of a set of finite-state automata that communicate with each other over FIFO channels.

We would now like to extend the MPA formalism to define automata which implement the timed partial order behaviours that we described. As a step towards this, we first look at finite-state automata with timing information which run over timed words. Timed automata, introduced in [5], are well-established models for real-time systems, in which normal finite-state automata are augumented with real-valued variables called *clocks*. A clock can be reset at any time and records the time elapsed since the last time it was reset. We are also interested in a subclass of timed automata, namely the event clock automata, which combine high expressivity with a tractable theory. An event clock automaton uses implicit "event clocks" that record or predict time lapses only with respect to the last or the next occurence of an event.

We introduce two models of message passing automata with timing. The first model we introduce is that of a timed message passing automaton. A timed MPA is an MPA equipped with clocks, as in a timed automaton. We can describe the global semantics of a timed MPA in terms of timed linearizations, which we then lift to obtain its semantics in terms of timed MSCs.

As a second and more robust model, we introduce event clock message passing automata, which combine the MPA formalism with that of the event clock automaton. We describe the semantics of event clock MPA directly over time-constrained MSCs and timed MSCs in a natural manner.

1.3.3 Specification formalisms

The language of mathematical logic is a classical formalism that allows us to specify properties of structures such as words and trees in a concise and precise manner. Here, we consider monadic second-order logic (MSO) as a specification language for MSCs. We then introduce additional timing predicates to obtain timed monadic second-order logic, allowing us to describe properties about timing in time-constrained MSCs and timed MSCs.

Another way to describe infinite collections of MSC behaviours is to use message sequence graphs (MSGs). These are finite-state automata whose nodes are labelled by MSCs. Each run of the MSG generates an MSC by concatenating the MSCs which are seen along the nodes encountered during the run. Instead, if we now consider an automaton in which nodes are labelled by time-constrained MSCs rather than MSCs, we obtain a time-constrained message sequence graph, which describes a possibly infinite collection of time-constained MSCs. The time-constrained MSG is thus an alternate global specification model for collections of time-constrained MSCs.

1.4 Other approaches and models for time and concurrency

Providing a timed partial order semantics as we have done above allows us to apply partial order reduction techniques [40] to address the model-checking problem. However, this is not the only way to handle time and concurrency issues in systems. Indeed, there are several other models that also handle time and concurrency in a comparable way.

Petri nets [55] are a widely used formalism to model concurrency in systems. In Petri nets, *tokens* are positioned in *places* and a transition fires by consuming tokens and creates new ones, in general in other places. Thus, transitions that consume different tokens, can fire independently. Many timed extensions of Petri nets have been considered, for instance, time Petri nets [14], timed Petri nets [54]. Unfoldings of Petri nets provide a way to model the partial order behaviour of these systems and by lifting these unfoldings to the timed extensions, they provide a timed partial order semantics [26]. For more discussion on this refer to [25]. However, these unfoldings are seldom graphically representable in a compact manner unlike MSCs (and their timed extensions). Further, unfoldings in Petri nets correspond to "branching time" whereas MSCs express "linear time" behaviour.

Other models dealing with time and concurrency include networks of timed automata [5] and products of timed automata [33]. Again in [18], unfolding techniques were applied to study such networks of timed automata. However, none of these models allow communication via explicit message passing, which is one of

the main features of the timed and event clock MPA that we have introduced.

The formal semantics and analysis of timing in MSCs has been addressed earlier in [7, 10, 24, 46]. In [7] and [10], only single timed MSCs or high-level timed MSCs were considered, while in [46] one of the first models of timed MPAs was introduced. However, the latter do not consider MSCs as a semantics of their automata but rather look at restricted channel architectures (e.g., one-channel systems) to transfer decidability of reachability problems from the untimed to the timed setting. The automaton model in [24] links the two approaches by considering a similar automaton model with semantics in terms of timed MSCs. They propose a practical solution to a very specific matching problem using the tool UPPAAL.

1.5 Results on message sequence charts with timing

Our first goal is to lift the Büchi-Elgot Theorem [20,37] to the timed and distributed setting. For the logical framework, we use the timed version of monadic second-order logic. We interpret both event clock MPAs and timed MSO formulae over timed MSCs and prove a constructive equivalence between them, with and without bounds on channels. This is done by lifting the corresponding results from the untimed case [17, 39, 42]. An important intermediary step in this translation is the reinterpretation of the timed MSO and event clock MPA in terms of time-constrained MSCs rather than timed MSCs. The time-constrained MSCs provide a dual link: they can be seen as MSCs whose labelings are extended by timing information and they can also be seen as a representation of infinite sets of timed MSCs. Once this translation is done, we can essentially follow the technique of [34] where such an equivalence is shown for timed words (without partial orders).

Next, we prove that, over *existentially bounded* channels, the problem of checking emptiness for our automaton model and thus, the satisfiability problem for our logic are decidable. Our approach consists of constructing a global finite timed automaton that can simulate the runs of an event clock MPA (which is a distributed machine) and so, reduce the problem to checking emptiness for a timed automaton. The hard part of the construction lies in "cleverly" maintaining the partial-order information (of the timed MSC) along the sequential runs of the global timed automaton, while using only finitely many clocks.

By adding time to MSCs, we encounter new questions and features which were absent in the untimed setting. For instance, time-constrained MSCs can be seen as specification models while timed MSCs are timed executions. Thus, we can think of each time-constrained MSC as being *realized* by the collection of timed MSCs which satisfy the timing constraints. This difference immediately suggests the following questions: If we are given a collection of time-constrained MSCs as a specification

and a timed MPA machine model with timed MSCs as its executions, does every time-constrained MSC in this collection have a witness timed MSC that realizes it? Similarly, is every timed MSC exhibited by the timed MPA witnessed by some time-constrained MSC in the specification? The problem becomes interesting when we consider infinite collections of time-constrained MSCs which can be represented by a time-constrained MSG. We first address the question of when such a time-constrained MSG describes a regular set of timed behaviours. Then, we are able to use this to provide solutions to the above mentioned problems.

1.6 An alternative model of time evolution in a distributed system

In the timed and distributed systems we have talked about so far, we have assumed that time evolves globally and all machines have access to the same clocks and same time. However, this is clearly not a general phenomenon. The internal computer clock of a laptop in a hot and dusty internet cafe in Chennai is likely to tick faster than the clock of a CVS server in an air-conditioned room in Paris. But we still need a way to describe how the laptop synchronizes with the CVS server in such a situation. In the last part of the thesis, we provide a framework for distributed systems with independently evolving local clocks. Each component of the system is modeled by a timed automaton. All clocks belonging to this timed automaton evolve at the same rate. However clocks belonging to different components are allowed to evolve at rates that are independent of each other. We allow clocks belonging to one component to be read/checked by another component but we require that a clock can only be reset by the component it belongs to.

Since we have unrelated time values on different components, we are interested in the underlying untimed behaviors of these distributed timed automata rather than their timed behaviors. Thus, the clocks (and time itself) are synchronization tools rather than being a part of the observation. This is a crucial point where our work departs from other existing works.

In [29, 53], for instance, classical timed automata are equipped with an additional parameter Δ, which allows a clock to diverge over a period t from its actual value by Δt. Also, in the model of [11], clocks are not shared and clocks on different processes drift only as long as the processes do not communicate. The model in [31] is even syntactically the same as ours since they allow clocks to be read across processes. However, in all these cases, the semantics are in terms of timed words rather than untimed languages, which is what we consider. This also explains why our automata differ from hybrid automata [43].

Another fundamental difference between all these approaches and our work is that we do not restrict to system configurations that can be reached under *some*

choice of local time rates. We will also tackle the problem of checking positive specifications by providing semantics that can check if a system exhibits some behavior under *all* relative clock speeds.

Indeed, it is natural to look at different semantics depending on the specifications that we want our system to satisfy. When we want to guarantee that our system exhibits a positive specification, we look at the *universal* semantics. This semantics describes the behaviors exhibited by the system no matter how time evolves in the individual components. However, if we want to check that our system avoids a negative specification, then we prefer to look at the *existential* semantics. This is the set of behaviors that the system might exhibit under some (bad) choice of local time rates in the components. We define a finite equivalence relation over the set of configurations of a distributed timed automaton using which we are able to show that the existential semantics always yields a regular set of untimed behaviors. Thus the model checking problem of distributed timed automata against regular negative specifications is decidable as well.

On the other hand, we show that checking both emptiness and universality is undecidable for the universal semantics. This is done by a reduction from Post's correspondence problem. This result is further strengthened to a bounded case, where we have restrictions on the relative time rates. Finally, to be able to synthesize and verify positive specifications, we introduce a more intuitive *reactive* semantics. In reactive semantics, we control the behaviour of the system, in a step-by-step manner, depending on how time progresses on each local component. Thus, we ensure that the resulting behaviours always satisfy a positive specification. By defining an equivalent alternating automaton, we are able to show that the reactive semantics always yields a regular set of behaviours.

1.7 Structure of the thesis

We start with some preliminary definitions and results in Chapter 2. In Section 2.1.2 we recall the notion of monadic second order logic over general structures. The rest of the chapter is split into two broad sections, the first dealing with the distributed setting and the second with timed setting. In Sections 2.2.1 and 2.2.2 we define MSCs and related notions regarding distributed behaviours. Next in Section 2.2.3 and 2.2.4 we define the automata and MSO model over MSCs respectively. This is followed by equivalence results over MSCs in Section 2.2.5. Finally in Section 2.2.6, we define message sequence graphs as a different approach to specify collections of MSCs. The second section of this chapter deals with timed behaviours in Section 2.3.1 followed by the timed automata in Section 2.3.2 and event clock automata in Section 2.3.3. We end the chapter by stating a logical characterization for event clock automata in Section 2.3.4.

The rest of the thesis is divided into two parts. The first deals with specifi-

cation and verification questions in the context of recording behaviours as timed partial orders. The second part of the thesis is devoted to the alternative model of distributed timed systems in which each component has a local notion of time.

In Chapter 3, we introduce the timed partial orders, namely timed MSCs and time-constrained MSCs and consider the systems that exhibit such behaviour namely, timed MPAs and event clock MPAs. Finally, we introduce the time-constrained MSG model to specify infinite collections of time-constrained MSCs.

In Chapter 4 we start by introducing the timed MSO logic and then prove the Büchi-Elgot Theorem in this timed partial order setting. This is followed by Chapter 5, where we show that the emptiness problem for event clock MPA model with existentially bounded channels is decidable. Finally Chapter 6 deals with time-constrained MSGs. We show how the global behaviour of a time-constrained MSG can be described as a timed automaton. In addition, we also relate time-constrained MSGs as a specification model to timed MPAs as an implementation model.

In the next part of the thesis, in Chapter 7, we introduce our distributed automaton model with independently evolving clocks, and define its existential and universal semantics. Section 7.2 extends the region construction for timed automata to our distributed setting, allowing us to compute a finite automaton recognizing the existential semantics. Section 7.3 shows that checking emptiness and universality of the universal semantics is undecidable. This result is sharpened towards bounded clock drifts in Section 7.4.1. Section 7.5 deals with the reactive semantics.

Finally, we conclude with some remarks and directions for future work in Chapter 8.

2
Preliminaries

2.1 Basic definitions

For a set Σ, we let Σ^* and Σ^ω denote the set of finite and, respectively, infinite words over Σ. The empty word is denoted by ε. We set $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ and $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. The concatenation of words $u \in \Sigma^*$ and $v \in \Sigma^\infty$ is denoted by $u \cdot v$. For a word $w \in \Sigma^*$, $\mathrm{prf}(w)$ denotes the set of prefixes of w and $|w|$ denotes the length of w, i.e, the number of symbols occuring in it with $|\varepsilon| = 0$. By a *word language over* Σ we mean a subset of Σ^*. An *alphabet* is a non-empty finite set. Given an alphabet Σ, we denote by Σ_ε the set $\Sigma \cup \{\varepsilon\}$.

The set of real numbers, rational numbers and integers are respectively denoted by $\mathbb{R}$, $\mathbb{Q}$ and $\mathbb{N}$. Further $\mathbb{R}_{\geq 0}$ and $\mathbb{R}_{>0}$ denote respectively non-negative and positive reals with corresponding notations for rationals and integers as well. For $t \in \mathbb{R}_{\geq 0}$, $\lfloor t \rfloor$ and $\mathit{fract}(t)$ refer, respectively, to the integral and fractional parts of t, hence, $t = \lfloor t \rfloor + \mathit{fract}(t)$.

Let $\mathcal{I}$ denote the set of all rationally bounded intervals over the real line. An interval is a non-empty, convex subset of non-negative reals. Thus, the intervals in $\mathcal{I}$ are of the form: *open* $(\ell, r) = \{m \in \mathbb{R} \mid \ell < m < r\}$ for $\ell, r \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$, *closed* $[\ell, r] = \{m \in \mathbb{R} \mid \ell \leq m \leq r\}$ for $\ell, r \in \mathbb{Q}_{\geq 0}$, *half-open* or *half-closed* $(\ell, r] = \{m \in \mathbb{R} \mid \ell < m \leq r\}$ for $\ell \in \mathbb{Q}_{\geq 0}, r \in \mathbb{Q}_{>0}$, $[\ell, r) = \{m \in \mathbb{R} \mid \ell \leq m < r\}$ for $\ell \in \mathbb{Q}_{\geq 0}, r \in \mathbb{Q}_{\geq 0} \cup \{\infty\}$. An interval set is a subset of $\mathcal{I}$.

A *(binary) relation* R on set S is a set of pairs of elements of S, i.e, a subset of $S \times S$. We write "$a\ R\ b$" to denote that (a, b) is a pair in R. The relation R is said to be a *partial order* if R is *reflexive* ($a\ R\ a$ for all $a \in S$), *antisymmetric* ($a\ R\ b$, $b\ R\ a$ implies $a = b$ for all $a, b \in S$) and *transitive* ($a\ R\ b$, $b\ R\ c$ implies $a\ R\ c$ for all $a, b, c \in S$). It is called a *total order* if in addition for any pair of elements $a, b \in S$ either the pair (a, b) or (b, a) belongs to R. A *linear extension* of a partial order R is a total order R' which contains all the pairs of R. Note that if $S = \{s_1, ..., s_n\}$ is a finite set and R is a total order then we can represent S as a sequence $(s_{i_1}, ..., s_{i_n})$ such that $s_{i_1}\ R\ s_{i_2}\ ...\ s_{i_{n-1}}\ R\ s_{i_n}$. In this case, s_{i_1} and s_{i_n} are respectively called the *minimal* and *maximal* element of S with respect to R. The *cardinality* or the number of elements in a finite set S is denoted $|S|$.

2.1.1 Finite-state automata over words

In this section, we recall the notion of finite-state automata, which are the standard implementation model for describing sequential behaviours, i.e word languages.

Definition 2.1. *Let Σ be an alphabet. A* finite-state automaton (FA) *over Σ is a tuple $\mathcal{C} = (Q, \delta, s_0, F_0)$, where*

- *Q is a non-empty finite set of states*
- *$\delta \subseteq Q \times \Sigma \times Q$ is the transition relation*
- *$s_0 \in Q$ is an initial state*
- *$F \subseteq Q$ is a set of final states*

The finite-state automaton is said to be deterministic (DFA) *if, for all $s, s_1, s_2 \in Q$, $a \in \Sigma$, $\{(s, a, s_1), (s, a, s_2)\} \subseteq \delta$ implies $s_1 = s_2$.*

A *run* r of a (deterministic) finite-state automaton $\mathcal{C}$ on a word $w = (a_1, \ldots, a_n) \in \Sigma^*$ is defined to be the sequence:

$$s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} s_n$$

where, for all $i \in \{1, \ldots, n\}$, $s_i \in Q$ and $(s_{i-1}, a, s_i) \in \delta$. The run is said to be *accepting* if $s_n \in F$. Note that, if $\mathcal{C}$ is deterministic then it has at most one run over a word from Σ^*. The language of $\mathcal{C}$, denoted $\mathcal{L}(\mathcal{C})$, is the set $\{w \in \Sigma^* \mid$ there exists an accepting run of $\mathcal{C}$ on $w\}$.

We sometimes view the run defined above to be the map $r : \{0, 1, \ldots, n\} \to Q$ such that for each $i \in \{0, \ldots, n\}$, $r(i) = s_i$ (in other words, $r(0) = s_0$ and for all $i \in \{1, \ldots, n\}$, $(r(i-1), a, r(i)) \in \delta$). This alternative presentation will be useful when we move from the sequential to the distributed setting, as we will see soon.

2.1.2 Monadic second-order logic over labeled relational structures

We recall the notion of monadic second-order logic (MSO) over extremely general structures, so that we can carry it over to words, MSCs and so on in the following sections and chapters. The structures that we define are over finite sets of elements that we refer to as *events*.

Now let us fix a finite set of symbols $\mathfrak{R}$ called the *relational symbols*. Each symbol $R \in \mathfrak{R}$ is interpreted as a relation on the set of events of a given structure.

Definition 2.2. *Given an alphabet* Σ*, a* labeled relational structure *over* Σ *is a tuple* $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda)$ *where* E *is a finite set of* events*,* $\mathfrak{R}^{\mathfrak{A}} = \{R^{\mathfrak{A}} \subseteq E \times E \mid R \in \mathfrak{R}\}$ *is a set of binary relations on* E *and* $\lambda : E \to \Sigma$ *is the* labeling function*.*

Example 2. For instance, a word $w \in \Sigma^*$ is such a structure $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda)$, where the set of events E is the set of positions $\{1, \ldots, |w|\}$ of the word w and the set $\mathfrak{R}^{\mathfrak{A}}$ has a single relation $<^{\mathfrak{A}}$ (thus, $\mathfrak{R} = \{<\}$) which refers to the obvious ordering between the positions (as elements of $\mathbb{N}_{>0}$). Then, the labeling function assigns to each position the letter of Σ at that position, thus defining the word.

Definition 2.3. *Let* Σ *be an alphabet and* Σ' *be a non-empty set. Then an* Σ'-extended labeled relational structure *over* Σ *is a tuple* $(E, \mathfrak{R}^{\mathfrak{A}}, \lambda, \rho)$ *where* $(E, \mathfrak{R}^{\mathfrak{A}}, \lambda)$ *is a labeled relational structure over* Σ *and* $\rho : E \to \Sigma'$ *is an* extended labeling function*.*

Example 3. A motivating example for this definition comes from *timed words* which will be studied in detail later. A (finite) *timed word* over Σ is a word $\sigma \in (\Sigma \times \mathbb{R}_{\geq 0})^*$, such that if $\sigma = (a_1, t_1) \ldots (a_n, t_n)$ for some $n \in \mathbb{N}_{>0}$ then $t_{i-1} \leq t_i$ for all $i \in \{2, \ldots, n\}$. Then, such a timed word can be seen as a $\mathbb{R}_{\geq 0}$-extended labeled relational structure over Σ where, as for words, the events are the positions $\{1, \ldots, n\}$ with the natural ordering between them and we have $\lambda(i) = a_i$, $\rho(i) = t_i$.

Now, we define the *monadic second-order logic* and its interpretation over the structures defined above.

Definition 2.4. *Let* Σ *be an alphabet. We denote* individual variables *by* $x, y, \ldots$ *and* set variables *by* $X, Y, \ldots$*. These variables will range over elements or sets of elements of* E*, respectively. We also use the predicates* $P_a(x)$*, where* x *is a variable,* $a \in \Sigma$*. Also, for a finite set of relational symbols* $\mathfrak{R}$*, we use binary predicates* $R(x, y)$ *where* x, y *are variables and* $R \in \mathfrak{R}$*. Then, the set* $\mathrm{MSO}(\Sigma, \mathfrak{R})$ of all monadic second-order logic formulae *over* Σ *with the set of relational symbols* $\mathfrak{R}$ *is generated inductively using the following grammar:*

$$\varphi \ ::= \ P_a(x) \mid x \in X \mid x = y \mid R(x, y) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi$$

We will use the usual abbreviations, for instance, for a variable x, we write $\forall x$ to mean $\neg\exists x\neg$. Also $\varphi_1 \to \varphi_2$ stands for $\neg\varphi_1 \vee \varphi_2$ and so on.

Semantics in terms of labeled relational structures

A formula φ from the above logic is interpreted on a labeled relational structure $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda)$ over the alphabet Σ (or indeed, on a Σ'-extended labeled relational structure $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda, \rho)$ over Σ for some non-empty set Σ').

For this, we have an *interpretation* $\mathbb{I}$ which is a map assigning individual variables to elements of E and set variables to sets of elements of E. We define the *satisfaction relation* $\mathfrak{A}, \mathbb{I} \models \varphi$ inductively as follows:

$$\mathfrak{A}, \mathbb{I} \models P_a(x) \text{ if } \lambda(\mathbb{I}(x)) = a \text{ for } a \in \Sigma \tag{2.1}$$
$$\mathfrak{A}, \mathbb{I} \models x \in X \text{ if } \mathbb{I}(x) \in \mathbb{I}(X) \tag{2.2}$$
$$\mathfrak{A}, \mathbb{I} \models x = y \text{ if } \mathbb{I}(x) = \mathbb{I}(y) \tag{2.3}$$
$$\mathfrak{A}, \mathbb{I} \models R(x, y) \text{ if } \mathbb{I}(x)\ R^{\mathfrak{A}}\ \mathbb{I}(y) \tag{2.4}$$
$$\mathfrak{A}, \mathbb{I} \models \neg\varphi \text{ if } \mathfrak{A}, \mathbb{I} \not\models \varphi \tag{2.5}$$
$$\mathfrak{A}, \mathbb{I} \models \varphi_1 \vee \varphi_2 \text{ if } \mathfrak{A}, \mathbb{I} \models \varphi_1 \text{ or } \mathfrak{A}, \mathbb{I} \models \varphi_2 \tag{2.6}$$
$$\mathfrak{A}, \mathbb{I} \models \exists x\varphi \text{ if } \exists e \in E \text{ such that } \mathfrak{A}, \mathbb{I}[x \mapsto e] \models \varphi \tag{2.7}$$
$$\mathfrak{A}, \mathbb{I} \models \exists X\varphi \text{ if } \exists E' \subseteq E \text{ such that } \mathfrak{A}, \mathbb{I}[X \mapsto E'] \models \varphi. \tag{2.8}$$

For sentences φ in this logic (i.e, those that have no free variables), we write $\mathfrak{A} \models \varphi$ instead of $\mathfrak{A}, \mathbb{I} \models \varphi$. Note that here and henceforth $\mathbb{I}[x \to e]$ for $e \in E$ denotes the interpretation $\mathbb{I}'$ such that $\mathbb{I}'(x) = e$ and for all variables $y \neq x$ and set variables X, $\mathbb{I}'(y) = \mathbb{I}(y), \mathbb{I}'(X) = \mathbb{I}(X)$. Similarly $\mathbb{I}[X \to E']$ for $E' \subseteq E$ is the interpretation which maps X to E' but coincides with $\mathbb{I}$ otherwise.

The *existential* fragment of $\mathrm{MSO}(\Sigma, \mathfrak{R})$, denoted $EMSO(\Sigma, \mathfrak{R})$, comprises of all formulae $\exists X_1 \ldots \exists X_n \varphi$ such that φ does not contain any set quantifier.

Example 4. It is now immediate to see that we can obtain the monadic second-order logic over words. This is done by just interpreting events as positions in the word as in Example 2.

For instance, the MSO formula $\forall X \forall x (x \in X) \to P_a(x)$ asserts that in every subset of positions, every position in the subset has the letter a. Thus, a word satisfies this formula if and only every position of the word is the letter a.

Thus, we get words over Σ as labeled relational structures and thus we have the definition of the logic $\mathrm{MSO}(\Sigma, \{<\})$ with semantics as words over Σ . We will just write $\mathrm{MSO}(\Sigma)$ to denote this logic.

For the sake of convenience and readability, we make the following *important* notational simplifications:

- When there is no scope for confusion about the structure $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda)$, we identify each relational symbol $R \in \mathfrak{R}$ with the relation $R^{\mathfrak{A}}$ on E and thus write $\mathfrak{A} = (E, \mathfrak{R}, \lambda)$ instead.
- If a labeled relational structure contains only one relation, i.e, it is of the form $\mathfrak{A} = (E, \{R^{\mathfrak{A}}\}, \lambda)$ for some $R^{\mathfrak{A}} \subseteq E \times E$, we write $\mathfrak{A} = (E, R, \lambda)$ instead.
- When they are clear from the context, we omit the parameters $(\Sigma, \mathfrak{R})$ and talk of MSO and EMSO.

The above mentioned abuses of notation also apply in the case of Σ'-extended labeled relational structures.

2.2 Distributed setting

Labeled partial orders For an alphabet Σ, a Σ-*labeled poset* is a labeled relational structure $(E, \leq, \lambda)$ over Σ, where $\leq$ is a partial order on the set of events E called its *ordering relation*.

A *linearization* of a Σ-labeled poset $(E, \leq, \lambda)$ is any Σ-labeled poset $(E, \leq', \lambda)$ such that $\leq'$ is a linear extension of $\leq$. Then, the set of events $E = \{e_1, \ldots, e_n\}$ can be rewritten as a sequence $e_{i_1} \leq' e_{i_2} \ldots \leq' e_{i_n}$ such that, $\lambda(e_{i_1}) \ldots \lambda(e_{i_n}) \in \Sigma^*$. Thus, any linearization of $(E, \leq, \lambda)$ can be identified with a unique word over Σ.

2.2.1 Message sequence charts

Let $Proc = \{p, q, r, \ldots\}$ be a non-empty finite set of *processes (agents)* that communicate through messages via reliable FIFO channels using an alphabet of *message types* $\mathcal{M}$. For $p \in Proc$, let $Act_p = \{p!q(m), p?q(m) \mid q \in Proc, q \neq p, m \in \mathcal{M}\}$ be the set of *communication actions of process* p . The action $p!q(m)$ is read as p *sends the message* m *to* q and the action $p?q(m)$ is read as p *receives the message* m *from* q. We set $Act = \bigcup_{p \in Proc} Act_p$. We also denote the set of *channels* by $Ch = \{(p, q) \in Proc \times Proc \mid p \neq q\}$.

Let $M = (E, \leq, \lambda)$ be an Act-labeled partial order. For $e \in E$, let $\downarrow e = \{e' \in E \mid e' \leq e\}$. For $X \subseteq E$, $\downarrow X = \bigcup_{e \in X} \downarrow e$. We call $X \subseteq E$ a *prefix* of M if $X = \downarrow X$. For $p \in Proc$ and $a \in Act$, we set $E_p = \{e \in E \mid \lambda(e) \in Act_p\}$ to be the set of all p-events and $E_a = \{e \in E \mid \lambda(e) = a\}$ to be the set of a-events.

For each $(p, q) \in Ch$, we define a relation $<_{pq}$ as follows, to capture the fact that channels are FIFO with respect to each message—if $e <_{pq} e'$, the message m read by q at e' is the one sent by p at e.

$$e <_{pq} e' \stackrel{\triangle}{=} \lambda(e) = p!q(m),\ \lambda(e') = q?p(m) \text{ and } |\downarrow e \cap E_{p!q(m)}| = |\downarrow e' \cap E_{q?p(m)}|$$

Finally, for each $p \in Proc$, we define the relation $\leq_{pp} = (E_p \times E_p) \cap \leq$, with $<_{pp}$ standing for the largest irreflexive subset of $\leq_{pp}$. Also, $\lessdot_{pp}$ denotes the *immediate successor relation* on process p: for $e, e' \in E_p$, $e \lessdot_{pp} e'$ if $e <_{pp} e'$ and for all $e'' \in E_p$, we have $e <_{pp} e'' \leq_{pp} e'$ implies $e'' = e'$.

Definition 2.5. *A* message sequence chart (MSC) *over Act is a* finite *Act-labelled*

poset $M = (E, \leq, \lambda)$ *that satisfies the following conditions,*

1. *Each relation* $\leq_{pp}$ *is a total order on* E_p. (2.9)
2. *If* $p \neq q$ *then for each* $m \in \mathcal{M}, |E_{p!q(m)}| = |E_{q?p(m)}|$ (2.10)
3. *If* $e <_{pq} e'$, *then* $|{\downarrow}e \cap (\bigcup_{m \in \mathcal{M}} E_{p!q(m)})| = |{\downarrow}e' \cap (\bigcup_{m \in \mathcal{M}} E_{q?p(m)})|$ (2.11)
4. *The partial order* $\leq$ *is the reflexive, transitive closure of* $\bigcup_{p,q \in Proc} <_{pq}$ (2.12)

The second condition ensures that every message sent along a channel is received. The third condition says that every channel is FIFO across all messages. For an MSC $M = (E, \leq, \lambda)$ over Act, we let $\mathrm{lin}(M)$ denote its set of linearizations seen as words over the set of actions Act.

In diagrams, the events of an MSC are presented in *visual order*. The events of each process are arranged in a vertical line and messages are displayed as horizontal or downward-sloping directed edges.

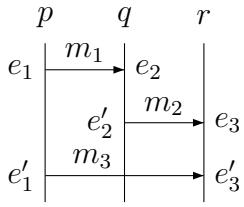


Figure 2.1: An MSC

Example 5. The Figure 2.1 shows an example of an MSC with three processes $\{p, q, r\}$ and six events $\{e_1, e'_1, e_2, e'_2, e_3, e'_3\}$ corresponding to three messages—m_1 from p to q, m_2 from q to r and m_3 from p to r. Then, for instance, $p!q(m_1)\ q?p(m_1)\ q!r(m_2)\ p!r(m_3)\ r?q(m_2)\ r?p(m_3)$ is one linearization or *execution* of this MSC seen as a word over Act.

Note that under the FIFO assumption an MSC can be reconstructed from any one of its linearizations. The relation $<_{pp}$ is determined by the order of p-events in the linearization, while $<_{pq}$ is determined by matching the i^{th} $p!q$-event in the linearization with the i^{th} $q?p$-event.

MSC languages An *MSC language over Act* is a set of MSCs over Act. We can also regard an MSC language $\mathcal{L}$ over Act, as a word language over Act consisting of all linearizations of the MSCs in $\mathcal{L}$. For an MSC language $\mathcal{L}$, we set $\mathrm{lin}(\mathcal{L}) = \bigcup\{\mathrm{lin}(M) \mid M \in \mathcal{L}\}$.

Definition 2.6. *An MSC language* $\mathcal{L}$ *is said to be a* regular MSC language *if the word language* $\mathrm{lin}(\mathcal{L})$ *is a regular language over Act.*

2.2.2 Boundedness in MSCs

An important subclass of MSCs is the set of *bounded* MSCs that correspond to systems whose channel capacity is restricted. These systems turn out to enjoy nice algorithmic properties and have liberal logical correspondances too.

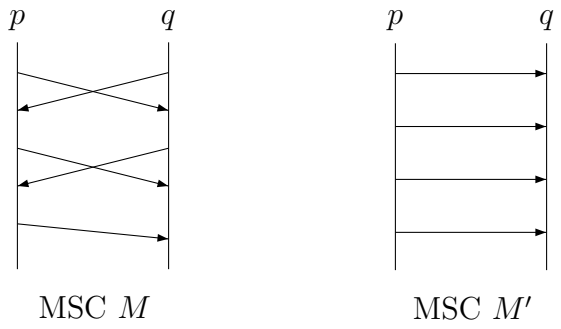


Figure 2.2: A ∀-2-bounded and a ∃-1-bounded MSC

Let $B \in \mathbb{N}_{>0}$ be a positive integer. Then, a word $w \in Act^*$ is said to be B-bounded if for any prefix u of w and any $p, q \in Proc$, the number of occurrences of $p!q$ exceeds the number of occurrences of $q?p$ by at most B. Thus,

Definition 2.7. *For an MSC M, $B \in \mathbb{N}_{>0}$, $w \in \text{lin}(M)$ is* B-bounded *if for every prefix v of w and every $(p,q) \in Ch$, $\sum_{m \in \mathcal{M}} |\pi_{\{p!q(m)\}}(v)| - \sum_{m \in \mathcal{M}} |\pi_{\{q?p(m)\}}(v)| \leq B$, where $\pi_\Gamma(v)$ denotes the projection of v on $\Gamma \subseteq Act$.*

This means that along the sequential execution of M described by w, no channel ever contains more than B-messages.

Definition 2.8. *An MSC M is called* universally B-bounded *(or ∀-B-bounded) if every $w \in \text{lin}(M)$ is B-bounded. It is said to be* existentially B-bounded *(or ∃-B-bounded) if there exists $w \in \text{lin}(M)$ such that w is B-bounded.*

Example 6. In the MSCs shown in Figure 2.2, we have abstracted away the message content and event-labeling for the sake of clarity. We see that the MSC M in Figure 2.2 is ∀-2-bounded, since along any execution, there are at most 2 messages in each channel. Further, the MSC M' is not ∀-1-bounded since during execution the process p could send two messages before q receives even one, i.e, there exists a linearization say $(p!q, p!q, p!q, p!q, q?p, q?p, q?p, q?p)$ for which the channel contains during execution more than one message. However there exists a linearization, namely $(p!q, q?p, p!q, q?p, p?q, q?p, p!q, q?p)$ which is 1 bounded, hence M' is ∃-1-bounded.

A set of MSCs is said to be ∃-B-*bounded (respectively,* ∀-B-*bounded)* if each MSC in the set is ∃-B-bounded (respectively, ∀-B-bounded). Further such a set is called *existentially bounded (respectively, universally bounded)* if there exists a B such that it is ∃-B-bounded (respectively, ∀-B-bounded). We have the following result from [42].

Lemma 2.9. *If an MSC language $\mathcal{L}$ is regular then it is ∀-B-bounded for some $B \in \mathbb{N}_{>0}$.*

2.2.3 Message passing automata

Message passing automata (also refered to as communicating finite-state machines) are a natural machine model for recognizing MSCs. We use the definition from [42]. Recall that Act denotes the set of actions or the communication alphabet over the set of processes $Proc$ and messages $\mathcal{M}$.

Definition 2.10. *A* message passing automaton (MPA) *over Act is a structure*

$$\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, \Delta, F)$$

where,

- Δ *is a finite alphabet of* auxiliary messages
- *For each $p \in Proc$, the component $\mathcal{A}_p$ is a structure $(S_p, \iota_p, \rightarrow_p)$, where:*
  - S_p *is a finite set of p-local states.*
  - $\iota_p \in S_p$ *is the p-local initial state.*
  - $\rightarrow_p \subseteq S_p \times Act_p \times \Delta \times S_p$ *is the p-local transition relation.*
- $F \subseteq \prod_{p \in Proc} S_p$ *is the finite set of global final states.*

The local transition relation $\rightarrow_p$ specifies how the process p sends and receives messages. The transition $(s, p!q(m), d, s')$ says that in state s, p can send the message m tagged with auxiliary data d to q and move to state s'. Similarly, the transition $(s, p?q(m), d, s')$ signifies that at state s, p can receive the message m tagged with the auxiliary data d from q and move to state s'.

Note that in the above definition, each message can be *tagged* with auxiliary data from the set Δ. The ability to convey this finite amount of extra auxiliary information turns out to be quite powerful, significantly increasing the expressive power of MPA. For further discussion on MPA without auxiliary data, refer to the *excellent* survey by Narayan Kumar [47].

Depending on whether we are interested in the distributed or the global behaviour, the semantics of message passing automata can be defined in two ways. We examine both these approaches in turn.

Semantics of MPAs over MSCs

In the style of [48], we can use MSCs directly to represent successful runs. In other words, an MPA will run over MSCs rather than linearizations of MSCs, allowing for its distributed behaviour.

To define the run of $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, \Delta, F)$ over an MSC $M = (E, \leq, \lambda)$, we first consider a function $r : E \rightarrow \bigcup_{p \in Proc} S_p$ which labels each event of E_p with a

local state from S_p. Now, we define $r^- : E \to \bigcup_{p \in Proc} S_p$ as follows: For $e \in E_p$, if there is another event $e' \in E_p$ such that $e' \lessdot_{pp} e$, then we set $r^-(e) = r(e')$. Otherwise (e is the minimal event of E_p with respect to $\leq_{pp}$), we set $r^-(e) = \iota_p$. Then, r is said to be a *run* of $\mathcal{A}$ on M if for all $e, e' \in E$ such that $e <_{pq} e'$, there exists $d \in \Delta$ such that,

$$(r^-(e), \lambda(e), d, r(e)) \in \rightarrow_p \text{ and } (r^-(e'), \lambda(e'), d, r(e')) \in \rightarrow_q .$$

We then define $f_p = r(e_p)$, where e_p is the maximal event of E_p with respect to $\leq_{pp}$ unless $E_p = \emptyset$, in which case we set $f_p = \iota_p$. Then run r is *successful* if the tuple $(f_p)_{p \in Proc} \in F$. An MSC is *accepted* by an MPA $\mathcal{A}$ if it admits a successful run. We denote by $\mathcal{L}_{MSC}(\mathcal{A})$, the set of MSCs that are accepted by $\mathcal{A}$.

Semantics of MPAs over linearizations of MSCs

In [42], a run of the MPA is defined on linearizations of MSCs rather than on MSCs, which reflects its operational behaviour at the expense that several execution sequences may get collapsed into a single run. Such a view relies on the global transition relation of the MPA as we detail below.

A global state of $\mathcal{A}$ is an element of $\prod_{p \in Proc} S_p$. For a global state $\overline{s}$, $\overline{s}_p$ denotes the p^{th} component of $\overline{s}$. A *configuration* is a pair $(\overline{s}, \chi)$ where $\overline{s}$ is a global state and $\chi : Ch \to (\mathcal{M} \times \Delta)^*$ is the *channel state* describing the message queue in each channel.

An *initial configuration* of $\mathcal{A}$ is of the form $(\overline{s}_{in}, \chi_\varepsilon)$ where $\overline{s}_{in} = (\iota_p)_{p \in Proc}$, $\chi_\varepsilon(c)$ is the empty string ε for every channel c. The set of *final configurations* of $\mathcal{A}$ is $F \times \{\chi_\varepsilon\}$.

The set of reachable configurations of $\mathcal{A}$, $Conf_{\mathcal{A}}$, is defined inductively, together with a transition relation $\Longrightarrow \subseteq Conf_{\mathcal{A}} \times Act \times Conf_{\mathcal{A}}$.

- $(\overline{s}_{in}, \chi_\varepsilon) \in Conf_{\mathcal{A}}$.
- Suppose $(\overline{s}, \chi) \in Conf_{\mathcal{A}}$, $(\overline{s}', \chi')$ is a configuration and there is a transition $(\overline{s}_p, p!q(m), d, \overline{s}'_p) \in \rightarrow_p$ such that for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi'((p,q)) = \chi((p,q)) \cdot (m, d)$, and for $c \neq (p,q)$, $\chi'(c) = \chi(c)$ then,

  there is a global move, $(\overline{s}, \chi) \stackrel{p!q(m)}{\Longrightarrow} (\overline{s}', \chi')$ and $(\overline{s}', \chi') \in Conf_{\mathcal{A}}$.
- Similarly, if $(\overline{s}, \chi) \in Conf_{\mathcal{A}}$, $(\overline{s}', \chi')$ is a configuration and there is transition $(\overline{s}_p, p?q(m), d, \overline{s}'_p) \in \rightarrow_p$ such that, for $r \neq p$, $\overline{s}_r = \overline{s}'_r$, $\chi((q,p)) = (m, d) \cdot \chi'((q,p))$, and for $c \neq (q,p)$, $\chi'(c) = \chi(c)$ then,

  there is a global move $(\overline{s}, \chi) \stackrel{p?q(m)}{\Longrightarrow} (\overline{s}', \chi')$ and $(\overline{s}', \chi') \in Conf_{\mathcal{A}}$.

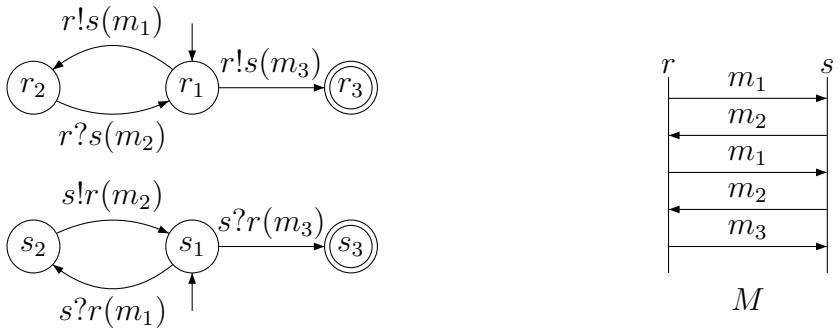


Figure 2.3: An MPA and an MSC recognized by it

For $w = (a_1 \dots a_n) \in Act^*$, a run of $\mathcal{A}$ over w is a map $r : \{0, \dots, n\} \to Conf_{\mathcal{A}}$ such that $r(0) = (\overline{s}_{in}, \chi_\varepsilon)$ and for each $i \in \{1, \dots, n\}$, $r(i-1) \xRightarrow{a_i} r(i)$. The run r is *accepting* if $r(n)$ is a final configuration. We define $\mathcal{L}(\mathcal{A}) = \{w \in Act^* \mid \mathcal{A}$ has an accepting run over $w\}$. It follows that, $\mathcal{L}(\mathcal{A})$ corresponds to the set of linearizations of a collection of MSCs.

Example 7. Figure 2.3 shows an MPA along with an MSC that it recognizes. In the MSC, we have omitted the event names/labels for clarity. In this MPA, at any point, r can choose to either send message m_3 to s and quit. Or it sends message m_1 and waits for an acknowledgement of message m_2 from s before continuing this loop. If Process s receives message m_3 it quits, else it sends message m_2 to r on receipt of message m_1. The adjoining MSC M shows a run scenario in which the loop sending message m_1 and acknowledgement m_2 occurs twice before message m_3 is sent by r. When it is received by s the run terminates at the global final state (r_3, s_3).

Note that each configuration of the MPA records the current contents of each channel, i.e, messages sent and as yet undelivered. A configuration $(\overline{s}, \chi)$ is said to be B-bounded for some $B \in \mathbb{N}$ if $|\chi(c)| \leq B$ for all $c \in Ch$. An MPA $\mathcal{A}$ is said to be *B-bounded* if every reachable configuration of $\mathcal{A}$ is B-bounded. Clearly, a B-bounded MPA accepts a universally B-bounded language of MSCs.

2.2.4 Monadic second-order logic over MSCs

Logic is a classical formalism to describe properties of various structures including words, trees, graphs etc. In this case we use it as a means to describe sets of MSCs. In particular we look at monadic second-order logic as way to describe these objects. We will in fact define several logics, which differ in their signature and whose syntax depends on the set $\mathfrak{R}$ of (binary) relation symbols, which we use to settle the access to the partial order relation of a given MSC.

First, we consider the singleton set of relation symbols containing just the partial order symbol, $\mathfrak{R}_{\leq} = \{\leq\}$. Then, by choosing $\Sigma = Act$ and $\mathfrak{R} = \mathfrak{R}_{\leq}$ in

Definition 2.4, we obtain the logic MSO$(Act, \mathfrak{R}_{\leq})$. Now, an MSC $M = (E, \leq, \lambda)$ over Act is formally a labeled relational structure $(E, \{\leq^M\}, \lambda)$ over Act, and thus we obtain the semantics of a formula of MSO$(Act, \mathfrak{R}_{\leq})$ in terms of MSCs over Act .

In addition, for an MSC M, we have the immediate successor relation $\lessdot_{pp}^M$ for each $p \in Proc$ and the message relation $<_{pq}^M$ for each $(p, q) \in Ch$. Thus, for $\mathfrak{R}_{\lessdot} = \{\lessdot_{pp} | \ p \in Proc\} \cup \{<_{pq} | \ p \neq q\}$ we obtain the variant of the logic MSO$(Act, \mathfrak{R}_{\lessdot})$ which talks only about the immediate partial order and its semantics in terms of MSCs over Act.

Again we have the *existential* fragments denoted EMSO$(Act, \mathfrak{R}_{\leq})$ (respectively, EMSO$(Act, \mathfrak{R}_{\lessdot})$) comprising of all formulas $\exists X_1 \ldots \exists X_n \varphi$ such that $\varphi \in$ MSO$(Act, \mathfrak{R}_{\leq})$ (respectively, MSO$(Act, \mathfrak{R}_{\lessdot})$) does not contain any set quantifier.

For a sentence φ in any of the logics defined above, we define $\mathcal{L}_{MSC}(\varphi)$ to be the set of all MSCs over Act which satisfy φ.

Example 8. Consider the MSO$(Act, \mathfrak{R}_{\leq})$ sentence φ given below,

$$\bigwedge_{(p,q)\in Ch} \forall x \forall y \forall z \Big(\big(\bigvee_{m\in\mathcal{M}} P_{p!q(m)}(x) \ \wedge \ \bigvee_{m\in\mathcal{M}} P_{p!q(m)}(y) \ \wedge \ \bigvee_{m\in\mathcal{M}} P_{p!q(m)}(z) \\ \wedge \ (x < y) \ \wedge \ (y < z)\big) \implies \exists x' ((x <_{pq} x') \wedge (x' < z)) \Big)$$

where indeed, $x < y$ denotes $x \leq y \wedge \neg(x = y)$ and so on. Then, we can observe that this sentence asserts in any sequence of 3 sends, the receive corresponding to the first send must occur before the third send. In other words, φ characterizes the set of all universally-2-bounded MSCs. Thus, if we take the MSCs M, M' shown in Figure 2.2, we have $M \models \varphi$ but $M' \not\models \varphi$.

2.2.5 Relations between logic and automata over MSCs

Büchi [20] and Elgot [37] show the relationship between MSO logic and finite state automata over words in one of the cornerstone results in automata theory. This result has been lifted to the MSC setting in different ways. Below we mention few of the relevant results.

In 2000, Henrikson et. al [42] gave a following characterization of regular MSC languages,

Theorem 2.11 (universally bounded MSCs, [42])**.** *Let $B \in \mathbb{N}_{\geq 0}$. Let $\mathcal{L}$ be a language of universally B-bounded MSCs. Then, the following are equivalent:*

1. *$\mathcal{L}$ is a regular MSC language.*
2. *$\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ for some sentence $\varphi \in$ MSO$(Act, \mathfrak{R}_{\leq})$.*

3. $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ *for some sentence* $\varphi \in \text{EMSO}(Act, \mathfrak{R}_{\leq})$.

4. $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{A})$ *for some* B*-bounded MPA* $\mathcal{A}$.

In [39], Genest, Kuske and Muscholl generalized this result to existentially bounded MSCs,

Theorem 2.12 (existentially bounded MSCs, [39])**.** *Let* $B \in \mathbb{N}_{\geq 0}$, $\mathcal{L}$ *be a language of existentially* B*-bounded MSCs. Then the following are equivalent:*

1. $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ *for some sentence* $\varphi \in \text{MSO}(Act, \mathfrak{R}_{\leq})$

2. $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ *for some sentence* $\varphi \in \text{EMSO}(Act, \mathfrak{R}_{\leq})$

3. $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{A})$ *for some MPA* $\mathcal{A}$.

In fact, for proving the above theorem, [39] first prove a crucial result which we state separately.

Proposition 2.13 (Proposition 5.14, [39])**.** *Let* $B \in \mathbb{N}_{\geq 0}$. *Then there exists a MPA* $\mathcal{A}$ *over Act such that* $\mathcal{L}_{MSC}(\mathcal{A})$ *is the set of all existentially* B*-bounded MSCs over Act.*

And using these two results, [39] show that over existentially bounded MSCs the logics $(E)\text{MSO}(Act, \mathfrak{R}_{<})$ and $(E)\text{MSO}(Act, \mathfrak{R}_{\leq})$ are equally expressively, i.e the *Msg* relation can be encoded using $\leq$.

Proposition 2.14 (Proposition 6.2, [39])**.** *Let* φ *be a* $(E)\text{MSO}(Act, \mathfrak{R}_{<})$ *formula and* $B \in \mathbb{N}_{\geq 0}$ *with* $\mathcal{L}_{MSC}(\varphi)$ *is existentially* B*-bounded. Then there exists a* $(E)\text{MSO}(Act, \mathfrak{R}_{\leq})$ *formula* ψ *such that* $\mathcal{L}_{MSC}(\psi) = \mathcal{L}_{MSC}(\varphi)$.

The above results show the equivalence of MSO logic and MPA automata when restricted to bounded classes of MSCs. The following *remarkable* result due to Bollig and Leucker [17] holds for arbitrary MSC languages. However the tradeoff is that the logic is weaker.

Theorem 2.15. *Let* $\mathcal{L}$ *be a language of MSCs. Then the following are equivalent:*

1. $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ *for some* $\varphi \in \text{EMSO}(Act, \mathfrak{R}_{<})$

2. $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{A})$ *for some MPA* $\mathcal{A}$.

2.2.6 Message sequence graphs

Message sequence graphs (MSGs) are yet another approach commonly used to specify infinite collections of MSCs. An MSG is a finite directed graph with designated initial and terminal vertices. Each vertex in an MSG is labelled by a non-trivial MSC, i.e, an MSC containing at least one event. The edges represent (asynchronous) MSC concatenation, in which one MSC is "pasted" below the other. We formalize these notions below.

Definition 2.16. *A* graph *is a tuple* $G = (V, \rightarrow, v_{in}, V_F)$ *where*

- V *is a set of vertices,*
- $v_{in} \in V$ *is the initial vertex*
- $\rightarrow \subseteq V \times V$
- $V_F \subseteq V$ *is a set of final vertices*

A *path* π through a graph G is a sequence $q_0 \rightarrow q_1 \rightarrow \cdots \rightarrow q_n$ such that $(q_{i-1}, q_i) \in \rightarrow$ for $i \in \{1, 2, \ldots, n\}$.

Definition 2.17. *A* message sequence graph (MSG) *, is a tuple* $\mathcal{G} = (G, \mathcal{L}^M, \Phi)$ *where*

- G *is a graph* $(V, \rightarrow, v_{in}, V_F)$*,*
- $\mathcal{L}^M$ *is a finite set of non-empty MSCs over Act.*
- $\Phi : V \rightarrow \mathcal{L}^M$ *is a map assigning an MSC to every vertex of the graph.*

For each vertex v of a given MSG $\mathcal{G} = (G, \mathcal{L}^M, \Phi)$, let $\Phi(v) \in \mathcal{L}^M$ be the MSC $M_v = (E^v, \leq^v, \lambda_v)$. We assume that the events are disjoint across each M_v. Now, for a non-empty path π in G, we define the *MSC generated by* π to be $M_\pi = (E^\pi, \leq^\pi, \lambda_\pi)$ 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 $e, e' \in E_p$ and either $\rho v \preceq \rho' v'$ or $\rho v = \rho' v'$ and in this case $e <^v_{pp} e'$.
  - $(e, \rho v) <^\pi_{pq} (e', \rho' v')$ for some processes $p \neq q$, if $\rho v = \rho' v'$ and $e <^v_{pq} e'$.

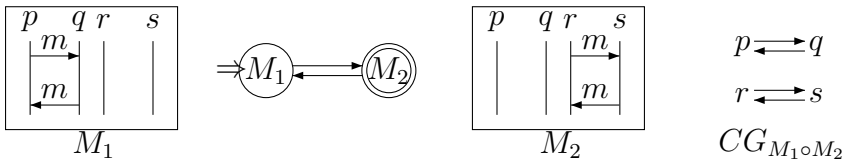


Figure 2.4: A message sequence graph

The MSC language of an MSG, $\mathcal{L}_{MSC}(\mathcal{G})$, is defined to be the set of all MSCs over Act generated by paths in the graph that start in an initial vertex and end in a final one. We say that an MSC language $\mathcal{L}$ is *MSG-definable* if there exists an MSG $\mathcal{G}$ such that $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{G})$.

Example 9. An example of an MSG is depicted in Fig. 2.4. The initial state is marked $\Rightarrow$ and the final state has a double circle. The language $\mathcal{L}$ defined by this MSG is *not* regular: $\mathrm{lin}(\mathcal{L})$ projected to $\{p!q(m), r!s(m)\}^*$ consists of $\sigma \in \{p!q(m), r!s(m)\}^*$ such that $|\pi_{p!q(m)}(\sigma)| = |\pi_{r!s(m)}(\sigma)| \geq 1$, which is not a regular string language.

Locally synchronized MSGs

In general, it is undecidable whether an MSG describes a regular MSC language [42]. However, a sufficient condition for the MSC language of an MSG to be regular is that the MSG be *locally synchronized*. We define this below, but first, we need the notion of a *communication graph* of an MSC.

Communication graph For an MSC $M = (E, \leq, \lambda)$, let CG_M, *the communication graph of* M, be the directed graph $(Proc, \mapsto)$ where:

- the set of nodes is the set of processes $Proc$ and
- the edge relation is: $(p, q) \in \mapsto$ iff there exists an $e \in E$ with $\lambda(e) = p!q(m)$.

M is said to be *com-connected* if CG_M consists of one nontrivial strongly connected component and isolated vertices.

Definition 2.18. *An MSG $\mathcal{G}$ is said to be* locally synchronized *[51] (or* bounded *[8]) if for every (simple) loop $\pi = q \to q_1 \to \cdots \to q_n \to q$, the MSC $\Phi(\pi)$ is com-connected.*

Example 10. In Figure 2.4, $CG_{M_1 \circ M_2}$ is not com-connected, so the MSG is not locally synchronized.

We have the following result for MSGs [8].

Theorem 2.19. *If $\mathcal{G}$ is locally synchronized MSG, then $\mathcal{L}_{MSC}(\mathcal{G})$ is a regular MSC language.*

2.3 Timed setting

In this section, we introduce the other building block of this thesis, namely the world of timed systems.

2.3.1 Timed words and languages

We study formal languages of *(finite) timed words*. We recall the definition which was already introduced in Example 3. A *timed word* over alphabet Σ is $\sigma \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ such that if $\sigma = (a_1, t_1)...(a_n, t_n)$ for some $n \in \mathbb{N}_{\geq 0}$, then the time-stamps t_i are non-decreasing, i.e, $t_{i-1} \leq t_i$ for all $i \in \{2, \ldots, n\}$. Let TW_Σ denote the set of all timed words over Σ.

Then, as observed in Example 3, a timed word over Σ can be viewed as a $\mathbb{R}_{\geq 0}$-extended labeled relational structure over Σ. For $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in \mathrm{TW}_\Sigma$, the associated extended labeled relational structure is $L_\sigma = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda, \rho)$ where, $E = \{1, \ldots, n\}$, $\mathfrak{R} = \{<\}$, with $<^E$ being the natural ordering over integers and $\lambda(i) = a_i, \rho(i) = t_i$.

A *timed language* over the alphabet Σ is a subset of TW_Σ. For a timed word σ, $\mathit{Untime}(\sigma)$ denotes the untimed word over Σ obtained by discarding the time stamps. For a timed language $\mathcal{L}$, $\mathit{Untime}(\mathcal{L}) = \{\mathit{Untime}(\sigma) \in \Sigma \mid \sigma \in \mathcal{L}\}$.

2.3.2 Timed automata

Alur and Dill [5] proposed a model of *timed automata* as an abstract model for real-time systems with finite control. Timed automata are finite-state machines whose transitions are constrained by timing requirements, so that they accept or generate timed words as behaviours. The finite control of the timed automaton consists of a finite set of states along with a finite set of real-valued variables called *clocks*. Each transition between states is allowed to *reset* some of the clocks. The value of each clock always records the amount of time that has elapsed since the last time the clock was reset. The transitions of the automaton specify arithmetic constraints on the clock values so that, during a run of the automaton, a transition may be taken only if the values of the clocks satisfy the corresponding constraints.

Example 11. Consider the timed automaton $\mathcal{B}_1$ shown in Figure 2.5. We can think of this automaton as modeling a "double-click" operation. At state s_0, if a click, i.e, action a, occurs twice quickly (within 2 time units), then we move to state s_2 and accept. Otherwise, after 2 time units have passed since our first a action, we are allowed to take the ε-transition back to state s_0 and try again.

To define this formally, we begin by defining the type of clock constraints that are allowed on the transitions. For a set of clocks $\mathcal{Z}$, the set $\mathrm{Form}(\mathcal{Z})$ of *clock*

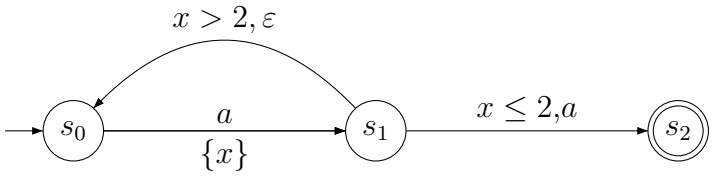


Figure 2.5: A timed automaton $\mathcal{B}_1$

constraints over $\mathcal{Z}$ is given by the grammar,

$$\varphi ::= \text{true} \mid \text{false} \mid x \bowtie c \mid \varphi \wedge \varphi \mid \varphi \vee \varphi$$

where x is a clock from $\mathcal{Z}$, $\bowtie \in \{<, \leq, >, \geq, =\}$, and c ranges over $\mathbb{Q}_{\geq 0}$.

A *clock valuation* over $\mathcal{Z}$ is a mapping $\nu : \mathcal{Z} \to \mathbb{R}_{\geq 0}$. We say that ν satisfies $\varphi \in \text{Form}(\mathcal{Z})$, written $\nu \models \varphi$, if φ evaluates to true using the values given by ν. Also, if $t \in \mathbb{R}_{>0}$, then we denote by $\nu + t$ the clock valuation such that $(\nu + t)(x) = \nu(x) + t$ for all $x \in \mathcal{Z}$. Finally, for $R \subseteq \mathcal{Z}$, $\nu[R \to 0]$ denotes the clock valuation defined by $\nu[R \to 0](x) = 0$ if $x \in R$ and $\nu[R \to 0](x) = \nu(x)$, otherwise.

Now, we are in a position to define the fundamental notion of timed automata as introduced in [5].

Definition 2.20. *A* timed automaton (with ε-transitions)*, denoted TA, is a tuple* $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, \mathit{Inv}, \iota, F)$ *where*

- S *is a set of* states*,*
- Σ *is the alphabet of* actions*,*
- $\mathcal{Z}$ *is a set of* clocks*,*
- $\delta \subseteq S \times \text{Form}(\mathcal{Z}) \times \Sigma_\varepsilon \times 2^{\mathcal{Z}} \times S$ *is the finite set of* transitions*,*
- $\mathit{Inv} : S \to \text{Form}(\mathcal{Z})$ *associates with each state an* invariant*,*
- $\iota \in S$ *is the* initial state*, and*
- $F \subseteq S$ *is the set of* final states*.*

Without loss of generality, we will assume that $\mathit{Inv}(\iota)$ is satisfied by the clock valuation over $\mathcal{Z}$ that maps each clock to 0. We let $\mathit{Reset}(\mathcal{B}) = \{x \in \mathcal{Z} \mid \text{there is } (s, \varphi, a, R, s') \in \delta \text{ such that } x \in R\}$ be the set of clocks that might be reset in $\mathcal{B}$. An *ε-transition* is a transition $(s, \varphi, a, R, s') \in \delta$ where $a = \varepsilon$. Further if $a = \varepsilon$ and

$R \neq \emptyset$, then such a transition is called an *ε-reset transition*. We call a TA $\mathcal{B}$ *finite* if S, $\mathcal{Z}$, and δ are finite.

A *run* of the timed automaton $\mathcal{B}$ is then defined as a sequence,

$$(s_0, \nu_0) \xrightarrow{a_1,t_1} (s_1, \nu_1) \xrightarrow{a_2,t_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n,t_n} (s_n, \nu_n)$$

where $n \geq 0$, $s_i \in S$, $a_i \in \Sigma_\varepsilon$, and $(t_i)_{1 \leq i \leq n}$ is a non-decreasing sequence of values from $\mathbb{R}_{\geq 0}$. Further, $\nu_i : \mathcal{Z} \to \mathbb{R}_{\geq 0}$ with $\nu_0(x) = 0$ for all $x \in \mathcal{Z}$. Finally, for all $i \in \{1, \ldots, n\}$, there are $\varphi_i \in \text{Form}(\mathcal{Z})$ and $R_i \subseteq \mathcal{Z}$ such that the following conditions hold:

$$(s_{i-1}, \varphi_i, a_i, R_i, s_i) \in \delta \tag{2.13}$$

$$\nu_{i-1} + t' - t_{i-1} \models Inv(s_{i-1}) \qquad \text{for each } t' \in [t_{i-1}, t_i] \tag{2.14}$$

$$\nu_{i-1} + t_i - t_{i-1} \models \varphi_i \tag{2.15}$$

$$\nu_i = (\nu_{i-1} + t_i - t_{i-1})[R_i \to 0] \tag{2.16}$$

$$\nu_i \models Inv(s_i) \tag{2.17}$$

We remark that the run of a TA can also be defined as an alternating sequence of *time-elapse* and *discrete* moves. In other words, each move in the above defined run $(s_{i-1}, \nu_{i-1}) \xrightarrow{a_i,t_i} (s_i, \nu_i)$ can split as:

- a *time-elapse move* of the form: $(s_{i-1}, \nu_{i-1}) \xrightarrow{\xi} (s_{i-1}, \nu_{i-1} + \xi)$ where $\xi = t_i - t_{i-1}$ and condition(2.14) holds,
- followed by a *discrete move* of the form: $(s_{i-1}, \nu_{i-1} + \xi) \xrightarrow{a_i} (s_i, \nu_i)$ where conditions (2.13), (2.15), (2.16) and (2.17) hold for some $\varphi_i \in \text{Form}(\mathcal{Z})$ and $R_i \subseteq \mathcal{Z}$.

The run is said to be *accepting* if $s_0 = \iota$ and $s_n \in F$. With each run as defined above, we can associate naturally the word $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in ((\Sigma \cup \{\varepsilon\}) \times \mathbb{R}_{\geq 0})^*$. Since an ε-transition is viewed as an invisible action, we remove from σ all pairs (a_i, t_i) such that $a_i = \varepsilon$. Thus we obtain a timed word $(a_{i_1}, t_{i_1}) \ldots (a_{i_{n'}}, t_{i_{n'}}) \in (\Sigma \times \mathbb{R}_{\geq 0})^*$ associated with every run. The timed language $\mathcal{L}_{tw}(\mathcal{B}) \subseteq \text{TW}_\Sigma$ is the set of timed words associated with accepting runs.

Two timed automata are said to be *equivalent* if the set of timed words accepted by them is the same. A timed language $\mathcal{L}$ is said to be *timed regular language* if $\mathcal{L} = \mathcal{L}_{tw}(\mathcal{B})$ for some timed automaton $\mathcal{B}$.

Example 12. Consider again, the timed automaton $\mathcal{B}_1$ shown in Figure 2.5, where $\Sigma = \{a\}$ and $\mathcal{Z} = \{x\}$. Then, only timed words over Σ that end with two occurences of a within 2 time units are accepted. For instance, the timed word $\sigma = (a, 1)(a, 5)(a, 7)$ is in $\mathcal{L}_{tw}(\mathcal{B}_1)$ as we have an accepting run of $\mathcal{B}_1$ on σ:

$$(s_0, x \mapsto 0) \xrightarrow{(a,1)} (s_1, x \mapsto 0) \xrightarrow{(\varepsilon,4)} (s_0, x \mapsto 3) \xrightarrow{(a,5)} (s_1, x \mapsto 0) \xrightarrow{(a,7)} (s_2, x \mapsto 2)$$

Note that this accepting run is not unique. On the other hand, we can observe that there is no accepting of $\mathcal{B}_1$ on $(a,1)(a,5)(a,8)$.

With these definitions and notations in place we are in a position to recall the following results about finite timed automata.

Theorem 2.21. *For a finite timed automaton $\mathcal{B}$,*

1. *Untime($\mathcal{L}_{tw}(\mathcal{B})$) is regular. Thus, the emptiness problem is decidable, i.e, given a finite TA $\mathcal{B}$, it is decidable to check if $\mathcal{L}_{tw}(\mathcal{B}) = \emptyset$ [5].*

2. *The universality problem, i.e, checking whether $\mathcal{B}$ accepts the set of all timed words is undecidable. Therefore, the inclusion problem is also undecidable, i.e, given finite TA $\mathcal{B}$ and $\mathcal{B}'$, it is undecidable to check if $\mathcal{L}_{tw}(\mathcal{B}) \subseteq \mathcal{L}_{tw}(\mathcal{B}')$ [5].*

3. *The class of timed regular languages is closed under (finite) union and intersection but not under complementation [5].*

4. *For a finite TA $\mathcal{B}$, if $\mathcal{B}$ has no ε-reset transitions, then we can construct an equivalent finite TA $\mathcal{B}'$ without ε-transitions [13].*

Though the theory of timed automata allows the solution of certain verification problems for real-valued timed systems, the general verification problem (i.e, language inclusion) is undecidable as stated above.

2.3.3 Event clock automata

The *event clock automata (ECA)* introduced in [6] are a subclass of timed automata which allow non-determinism and are closed under complementation. Further, in [34] D'Souza showed that these automata have a nice logical characterization via monadic second-order logic interpreted over timed words.

In a timed automaton, each clock records the time difference between the current input alphabet being read and the alphabet that was read when the clock was last reset. Thus, this association between clocks and the input alphabet read is dynamically determined by the behaviour of the automaton. In contrast, an ECA associates each clock to a fixed input letter. The idea is to interpret the input alphabet symbols as events or actions of the system and so, the event clock measures the time that has elapsed since the previous occurence of the event. In addition event-predicting clocks are also allowed, which measure the time *to* the next occurence of an event.

The standard way to present such an ECA as done in [6] uses two real-valued variables as clocks for each letter of the alphabet (to record and predict events) and defines a valuation of these clocks to reflect the input-regulated behaviour.

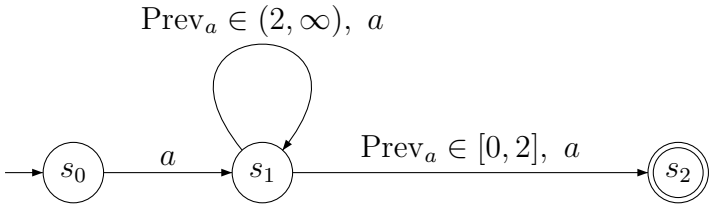


Figure 2.6: An event clock automaton

We adopt a slightly different approach which uses an equivalent but different presentation better suited for the results and generalizations that we will see in the forth-coming chapters.

Along a run of a TA, a clock guard verifies that the time elapsed between two positions of the timed word, satisfies some constraint. The two positions in question are the current position and the last position where the clock was reset. Now, in ECA, the relation between these two positions are independent of the automaton and determined by the input word itself. Thus, we can define the clock guards as constraints on these related positions. Then, an ECA can be seen as a finite state automaton running over timed words by using an extended alphabet consisting of these guards.

Example 13. Consider the event clock automaton shown in Figure 2.6 which again simulates the "double-click" operation as in Example 11. For an interval I, we write $\text{Prev}_a \in I$ on a transition to mean that the time elapsed since the previous occurence of a must be in the interval I. Thus, in Figure 2.6 the first a action sends us to state s_1. Now, if at the next a action, we find the previous one occured within 2 time units we go to state s_2 and accept. Otherwise we loop in state s_1. Similarly, we could formulate another simple automaton which uses next occurence in the guard instead of previous.

We formalize these ideas below. For an alphabet Σ, we fix a set of symbols $\text{EC} = \{\text{Prev}_a \mid a \in \Sigma\} \cup \{\text{Next}_a \mid a \in \Sigma\}$. These symbols are then interpreted as (binary) relations over the set of positions of a timed word. Given a timed word $\sigma = (a_1, t_1) \dots (a_n, t_n) \in \text{TW}_\Sigma$, recall (from Section 2.3.1) that it can be seen as a labeled relational structure $L_\sigma = (E = \{1, \dots, n\}, \{<\}, \lambda)$. Then,

- $\text{Prev}_a^\sigma = \{(i, j) \in E \times E \mid \lambda(j) = a,\ j < i,\ (k < i \wedge \lambda(k) = a) \implies k \leq j\}$
- $\text{Next}_a^\sigma = \{(i, j) \in E \times E \mid \lambda(j) = a,\ i < j,\ (i < k \wedge \lambda(k) = a) \implies j \leq k\}$

Let $[\text{EC} \dashrightarrow \mathcal{I}]$ denote the set of partial maps from the set of symbols EC to the set of intervals $\mathcal{I}$. Intuitively, interpreted over a timed word σ, $g \in [\text{EC} \dashrightarrow \mathcal{I}]$ is a map assigning an interval to some pairs related by Next_a^σ or Prev_a^σ.

Now, we are in a position to define the event clock automaton formally.

Definition 2.22. *An* event clock automaton (ECA) *is a tuple* $\mathcal{A} = (S, \Sigma, \delta, \iota, F)$ *where*

- S *is a finite set of states*
- Σ *is the alphabet*
- $\iota \in S$ *is the initial state*
- $F \subseteq S$ *is the set of final states*
- $\delta \subseteq (S \times [EC \dashrightarrow \mathcal{I}] \times \Sigma \times S)$ *is the finite set of transitions*

We will now define a run of the ECA on a timed word, as a map from the set of positions of the timed word to the set of states, rather than as a sequence of states (as was done in the previous section for timed automata). The reason we prefer this definition here will become clear in the next chapter, where we lift this definition easily to partial orders rather than words.

Let $\sigma = (a_1, t_1) \dots (a_n, t_n) \in \mathrm{TW}_\Sigma$ be a timed word with associated $\mathbb{R}_{\geq 0}$-extended labeled relational structure $L_\sigma = (\{1, \dots, n\}, \{<\}, \lambda, \rho)$ where $\lambda(i) = a_i, \rho(i) = t_i$. A *run* of ECA $\mathcal{A}$ over σ is a map $r : \{0, \dots, n\} \to S$ such that $r(0) = \iota$ and for each $i \in \{1, \dots, n\}$, there is a guard $g_i \in [\mathrm{EC} \dashrightarrow \mathcal{I}]$ such that:

- $(r(i-1), g_i, \lambda(i), r(i)) \in \delta$,
- for each symbol $\mathrm{Prev}_a \in \mathrm{dom}(g_i)$, there exists $j \in \{1, \dots, n\}$ such that $(i, j) \in \mathrm{Prev}_a^\sigma$ and $|\rho(i) - \rho(j)| \in g_i(\mathrm{Prev}_a)$ and
- for each symbol $\mathrm{Next}_a \in \mathrm{dom}(g_i)$, there exists $j \in \{1 \dots, n\}$ such that $(i, j) \in \mathrm{Next}_a^\sigma$ and $|\rho(j) - \rho(i)| \in g_i(\mathrm{Next}_a)$

The run r is said to be *successful* if $r(n) \in F$. A timed word is *accepted* by an ECA if it admits a successful run. We use $\mathcal{L}_{tw}(\mathcal{A})$ to denote the timed words that are accepted by an ECA $\mathcal{A}$.

Example 14. Again, consider the event clock automaton from Figure 2.6. Then, the timed word $\sigma = (a, 1)(a, 5)(a, 7)$ is in its timed language, since the mapping r defined by $r(1) = r(2) = s_1$, $r(3) = s_2$ is an accepting run. In fact, it can be seen that the timed language of this ECA is the same as the timed language of the timed automaton $\mathcal{B}_1$ from Figure 2.5.

2.3.4 Logical characterization of event clock automata

In this section we extend the monadic second-order logic to include timing predicates as done in [34]. The aim is to have a logic that can be interpreted over timed words and can characterize the class of languages recognized by the event clock automata of the previous section.

Here, again, we assume a supply of individual variables $x, y, \ldots$, and set variables $X, Y, \ldots$ which range over positions (and sets of positions) of a given timed word. Other than the usual predicate $P_a(x)$ for $a \in \Sigma$, we also have the event clock timing predicates $\text{Prev}_a(x) \in I$ and $\text{Next}_a(x) \in I$ where, x is an individual variable, $a \in \Sigma$ and $I \in \mathcal{I}$. Then,

Definition 2.23. *The set* ecMSO$(\Sigma, \{<\})$ *of all timed monadic second-order logic formulae over* Σ *with the relational symbol* $<$*, is generated inductively using the following grammar:*

$$\varphi \ ::= \ P_a(x) \mid x \in X \mid x < y \mid \text{Prev}_a(x) \in I \mid \text{Next}_a(x) \in I \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x \varphi \mid \exists X \varphi$$

Since, the ordering relation is obvious over timed words, we write ecMSO(Σ) instead of ecMSO$(\Sigma, \{<\})$. To define the semantics of a formula φ of this logic over timed words, we again associate a timed word $\sigma = (a_1, t_1) \ldots (a_n, t_n)$ with its $\mathbb{R}_{\geq 0}$-extended labeled relational structure $\mathfrak{A}_\sigma = (E = \{1, \ldots, n\}, \{<^{\mathfrak{A}}\}, \lambda, \rho)$. Then, the definition of interpretation $\mathbb{I}$ and the satisfaction relation $\models$ are the same as in the untimed case (refer Section 2.1.2), except for the timing predicates for which we have,

- $\mathfrak{A}_\sigma, \mathbb{I} \models \text{Prev}_a(x) \in I$ if $\exists j \in E$ s.t. $(\mathbb{I}(x), j) \in \text{Prev}_a^\sigma$ and $|\rho(\mathbb{I}(x)) - \rho(j)| \in I$,
- $\mathfrak{A}_\sigma, \mathbb{I} \models \text{Next}_a(x) \in I$ if $\exists j \in E$ s.t. $(\mathbb{I}(x), j) \in \text{Next}_a^\sigma$ and $|\rho(j) - \rho(\mathbb{I}(x))| \in I$.

Intuitively, the timing predicate $\text{Prev}_a(x) \in I$ asserts that the time elapsed since the last occurence of an a-action with respect to position $\mathbb{I}(x)$, lies in the interval I. Similarly, $\text{Next}_a(x) \in I$ asserts that the time that will elapse before the next occurence of an a-action (with respect to position x) will lie in the interval I.

The remaining notions are exactly as in the untimed case and thus, for a sentence φ of this logic, we define $\mathcal{L}_{tw}(\varphi) = \{\sigma \in \text{TW}_\Sigma \mid \mathfrak{A}_\sigma \models \varphi\}$.

Thus, we are in a position to state the main theorem proved by D'Souza in [34].

Theorem 2.24. *Let* $\mathcal{L} \subseteq \text{TW}_\Sigma$*. Then* $\mathcal{L} = \mathcal{L}_{tw}(\mathcal{A})$ *for some ECA* $\mathcal{A}$ *over* Σ *if and only if* $\mathcal{L} = \mathcal{L}_{tw}(\varphi)$ *for some sentence* $\varphi \in ecMSO(\Sigma)$*.*

Part I

Changing the behaviour

3

Adding Time to Scenarios and Systems

3.1 Timed message sequence charts

The first natural attempt while trying to add timing information to MSCs would be to add time stamps to the events of the MSCs. This is motivated from timed words where we have words with time stamps added at each letter (action). This approach is quite realistic when we want to model the real-time execution of concurrent systems.

We recall that *Proc* is a finite set of processes that communicate via FIFO channels using messages from the set $\mathcal{M}$. This generates the alphabet of communication actions $Act = \{p!q(m), p?q(m) \mid p, q \in Proc, q \neq p, m \in \mathcal{M}\}$.

Definition 3.1. *A* timed MSC *(TMSC) over Act is a pair* (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* $e, e' \in E$. *The set of all TMSCs over Act is denoted* $\mathbb{TMSC}(Act)$.

In the above definition note that over events e, e', $\leq$ is the partial order relating events, while over reals $t(e), t(e')$, $\leq$ refers to the usual ordering between real numbers.

Example 15. Consider the TMSC T presented in Figure 3.1 which shows a scenario where two user processes p and r interact with a railway ticket-booking server process q to book the last available ticket for a certain train journey. Actions are of the form $p!q(req)$ meaning that *User1* sends the *Server* a request for a ticket. In the scenario shown, both users send booking requests to the *Server* at time instant 1. The *Server* grants *User1*'s request since it is received first. Then, *User1* confirms his/her booking which leads to the server denying the booking to *User2*. Meanwhile, *User2* repeats his/her request at instant 5 which reaches the *Server* at time 8.

An *execution* or *linearization* of this scenario is:

$(p!q(req), 1)(r!q(req), 1)(q?p(req), 1)(q?r(req), 2)(q!p(grant), 3.5)(p?q(grant), 3.5)$

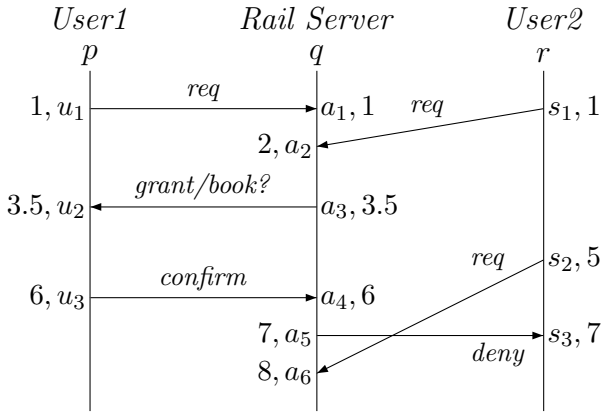


Figure 3.1: A timed MSC T describing interaction of two users with a server

$(r!p(req), 5)(p!q(conf), 6)(q?p(conf), 6)(q!r(deny), 7)(r?q(deny), 7)(q?r(req), 8)$

Note that in the above execution, the ordering on the time stamps is preserved, thus making it a timed word over the alphabet of actions. Such an execution is called a *timed linearization* of T. A single TMSC might have more than one timed linearization if concurrent events on different processes have the same time stamp. For instance, if we swap the first two pairs in the above execution we obtain another execution which respects the time stamps. Finally, observe that not all linearizations are timed linearizations as seen by swapping the last two pairs in the above execution. Formally,

Timed linearizations Let $T = (M, t)$ be a TMSC over Act with $M = (E, \leq, \lambda)$. Consider a linearization $(E, \leq', \lambda)$ of $(E, \leq, \lambda)$ according to which the events of E can be written as a sequence $e_1 \leq' e_2 \ldots \leq' e_n$. Then, the structure $(E, \leq' \lambda, t)$ is said to be a *linearization* of the TMSC T . If in addition, for all $1 \leq j < k \leq n$, we have $t(e_j) \leq t(e_k)$, then it is said to be a *timed linearization* . Indeed, this timed linearization can be seen as a timed word σ over Act by uniquely identifying it with $\sigma = (\lambda(e_1), t(e_1)) \ldots (\lambda(e_n), (t(e_n)) \in \mathrm{TW}_{Act}$. We let t-lin$(T) \subseteq \mathrm{TW}_{Act}$ denote the set of timed linearizations of a TMSC T, seen as a timed word language over Act.

As is the case with untimed MSCs, under the FIFO assumption for channels, a timed MSC can be faithfully reconstructed from any one of its timed linearizations.

TMSC languages A *TMSC language* $\mathcal{L}$ *over* Act is a set of TMSCs over Act. Then, t-lin$(\mathcal{L})$ is the timed word language over Act consisting of all timed linearizations of the TMSCs in $\mathcal{L}$, i.e, t-lin$(\mathcal{L}) = \bigcup\{\text{t-lin}(T) \mid T \in \mathcal{L}\}$.

3.1.1 Bounded channel setting

As in the case of untimed MSCs, restricting the channel capacity in TMSCs gives rise to an interesting, more "tractable" subclass of TMSCs. We extend the definition of existential and universal bounds from MSCs to TMSCs.

Definition 3.2. *A TMSC* (M, t) *is called* untimed-existentially-B-bounded ($\exists^u$-B-*bounded) if the MSC* M *is* $\exists$-B-*bounded. Similarly* (M, t) *is* untimed-universally-B-bounded ($\forall^u$-B-*bounded) if* M *is* $\forall$-B-*bounded.*

Note that directly lifting the definition of bounds from the untimed version does not preserve the timed linearizations. Consider the following example.

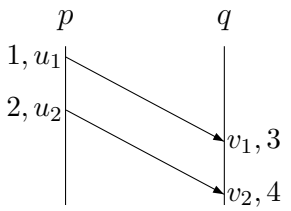


Figure 3.2: TMSC T'

Example 16. In the adjoining figure TMSC T' is $\exists^u$-1-bounded since there exists a linearization of the untimed MSC which is 1-bounded, namely, $u_1v_1u_2v_2$. However, this does not correspond to a timed linearization. In fact, the only timed linearization of T, namely $(u_1, 1)(u_2, 2)(v_1, 3)(v_2, 4)$ is 2-bounded. This example motivates us to consider an alternative definition of a timed notion of boundedness.

Definition 3.3. *For a TMSC* $T = (M, t)$, $B \in \mathbb{N}_{>0}$, *a timed linearization of* T, $(a_1, t_1) \dots (a_n, t_n) \in$ t-lin(T) *is said to be* B-bounded *if* $a_1 \dots a_n$ *is a* B-*bounded linearization of* M.

Definition 3.4. *A TMSC* T *is* timed-existentially-B-bounded ($\exists^t$-B-*bounded) if there exists a timed linearization of* T *which is* B-*bounded. Similarly* T *is* timed-universally-B-bounded ($\forall^t$-B-*bounded) if every timed linearization of* T *is* B-*bounded.*

Then, the following lemma is a straightforward consequence of the definitions.

Lemma 3.5. *If a TMSC* T *is* $\exists^t$-B-*bounded for some* $B > 0$, *then* T *is also* $\exists^u$-B-*bounded. Similarly, if* T *is* $\forall^u$-B-*bounded then it is also* $\forall^t$-B-*bounded.*

3.2 Timed message passing automata

In this section, we define a basic model of a timed and distributed system that runs on timed message sequence charts. The *timed message passing automata* are a combination of timed automata over timed words (from Section 2.3.2) and message passing automata over MSCs (from Section 2.2.3).

Thus, we begin with the set of communication actions Act over the set of processes $Proc$ and message alphabet $\mathcal{M}$ as before. In addition, we have a set

of clocks $\mathcal{Z}$ which is partitioned into the set of clocks belonging to each process, i.e, $\mathcal{Z} = \bigcup_{p \in Proc} \mathcal{Z}_p$. Each clock constraint is of the form $\varphi \in \text{Form}(\mathcal{Z})$ for some $p \in Proc$. As before, a clock valuation is a map $\nu : \mathcal{Z} \to \mathbb{R}_{\geq 0}$ and we write $\nu \models \varphi$, if φ evaluates to true using the values given by ν.

Definition 3.6. *A* timed message passing automaton (TMPA) *over Act with a set of clocks $\mathcal{Z}$ is a structure $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, \Delta, F, \mathcal{Z})$ where,*

- *Δ is a finite set of auxiliary messages*
- *For each $p \in Proc$, the component $\mathcal{A}_p$ is a structure $(S_p, \iota_p, \to_p, Inv_p)$, where:*
  - *i. S_p is a finite set of p-local states.*
  - *ii. $\iota_p \subseteq S_p$, is a set of initial states for p.*
  - *iii. $\to_p \subseteq S_p \times Form(\mathcal{Z}) \times Act_p \times \Delta \times 2^{\mathcal{Z}} \times S_p$ is the p-local transition relation.*
  - *iv. $Inv_p : S_p \to Form(\mathcal{Z})$ assigns an* invariant *to each p-local state.*
- *$F \subseteq \prod_{p \in Proc} S_p$ is the finite set of global final states.*

Further, $\mathcal{A}$ is said to be locally timed *if the guards, resets and invariants in $\mathcal{A}_p$ only refer to clocks from $\mathcal{Z}_p$, i.e., Conditions iii.,iv. above are respectively replaced by $\to_p \subseteq S_p \times Form(\mathcal{Z}_p) \times Act_p \times \Delta \times 2^{\mathcal{Z}_p} \times S_p$ and $Inv_p : S_p \to Form(\mathcal{Z}_p)$.*

As in the untimed case, each message can be *tagged* with auxiliary data from the set Δ. The transition $(s, \varphi, p!q(m), d, R, s')$ says that in state s, p can send the message m tagged with auxiliary data d to q and move to state s'. This transition is *guarded* by the clock constraint φ, i.e., the transition is enabled only when the current values of all the clocks satisfy φ. The set R specifies the clocks whose values are reset to 0 when this transition is taken. Similarly, the transition $(s, \varphi, p?q(m), d, R, s')$ signifies that at state s, p can receive the message m tagged with auxiliary data d from q and move to state s' provided the current clock values satisfy φ. Once again, all clocks in R are reset to 0.

Example 17. Figure 3.3 shows a timed MPA along with two of the timed MSCs that it recognizes. In the timed MSCs, we have only written the time-stamps associated with the events and not the event names themselves. In this timed MPA, r sends a message m_1 to s. Process s replies with m_2 exactly 1 time unit after it receives m_1. If m_2 is received by r within 2 time units of its sending m_1, it sends m_3 and quits. Otherwise, if at least 2.2 time units go by before r receives m_2, it resends m_1. Note that there is no transition enabled in r for the interval $2 < x \leq 2.2$. We may also observe here that this TMPA is locally timed since each process r, s only uses its clocks x, y respectively.

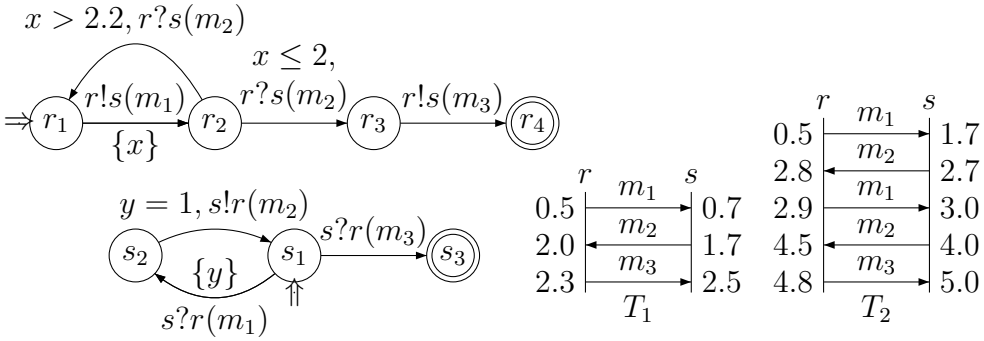


Figure 3.3: A timed MPA and some timed MSCs that it recognizes

To formally define the operational semantics of a general TMPA, we can consider it to be a global timed system running over timed linearizations of TMSCs. Thus, given a TMPA $\mathcal{A}$, we define its global timed automaton $\mathcal{B}_\mathcal{A}$ as follows.

The states of the global timed automaton are the configurations of the untimed MPA underlying $\mathcal{A}$. That is, a state of $\mathcal{B}_\mathcal{A}$ is a pair $(\overline{s}, \chi)$ where $\overline{s}$ is a global state of $\mathcal{A}$ and $\chi : Ch \to (\mathcal{M} \times \Delta)^*$ is the *channel state* describing the message queue in each channel c. Recall that a global state of $\mathcal{A}$ is an element of $\prod_{p \in Proc} S_p$ and for a global state $\overline{s}$, $\overline{s}_p$ denotes the p^{th} component of $\overline{s}$. Again, the initial states of $\mathcal{B}_\mathcal{A}$ are states of the form $(\overline{s}_{in}, \chi_\varepsilon)$ where $\overline{s}_{in} \in \prod_{p \in Proc} \iota_p$, $\chi_\varepsilon(c)$ is the empty string ε for every channel c. The set of final states of $\mathcal{B}_\mathcal{A}$ is the set $F \times \{\chi_\varepsilon\}$.

Now, the transition relation $\delta_{\mathcal{B}_\mathcal{A}}$ of $\mathcal{B}_\mathcal{A}$ is defined as follows:

$$((\overline{s}, \chi), \varphi, a, d, R, (\overline{s}', \chi')) \in \delta_{\mathcal{B}_\mathcal{A}}$$

if the following hold:

- for p such that $a \in Act_p$, $(\overline{s}_p, \varphi, a, d, R, \overline{s}'_p) \in \to_p$ and for $q \neq p, \overline{s}_q = \overline{s}'_q$,
- if $a = p!q(m)$, for some p, q, m, then $\chi'((p,q)) = \chi((p,q)) \cdot (m, d)$, and for $c \neq (p,q)$, $\chi'(c) = \chi(c)$.

Thus, $\mathcal{B}_\mathcal{A}$ is a timed automaton and we define the timed word language of a TMPA $\mathcal{A}$ to be the timed language of its global timed automaton, i.e, $\mathcal{L}_{tw}(\mathcal{A}) = \mathcal{L}_{tw}(\mathcal{B}_\mathcal{A})$. Then, we may observe that every timed word in $\mathcal{L}_{tw}(\mathcal{A})$ is a timed linearization of some TMSC over Act. This allows us to define the TMSC language of a TMPA $\mathcal{A}$, denoted by $\mathcal{L}_{time}(\mathcal{A})$, to be the set of TMSCs over Act that have some timed linearization in $\mathcal{L}_{tw}(\mathcal{A})$.

One may note, however, that this is not the only way to define the TMSC language of a TMPA. For instance, we could require that all the timed linearizations of a TMSC should be accepted by the TMPA for the TMSC to be in the language. This gives us a different definition: $\mathcal{L}^{\forall}_{time}(\mathcal{A}) = \{T \in \mathbb{TMSC}(Act) \mid \text{for all } \sigma \in \text{t-lin}(T), \sigma \in \mathcal{L}_{tw}(\mathcal{A})\}$. Then, we observe that $\text{t-lin}(\mathcal{L}^{\forall}_{time}(\mathcal{A})) \subseteq \mathcal{L}_{tw}(\mathcal{A}) \subseteq$

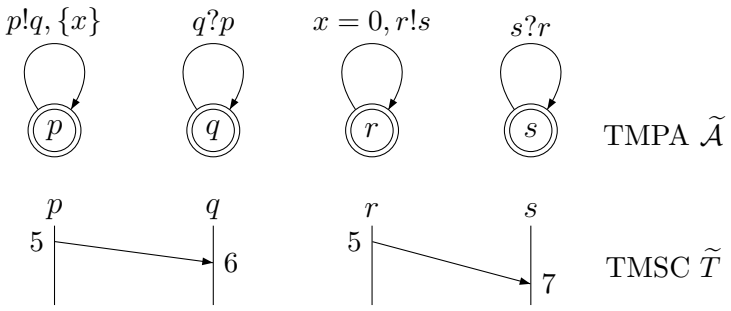


Figure 3.4: An example of a TMPA $\widetilde{\mathcal{A}}$ that is not locally timed

t-lin($\mathcal{L}_{time}(\mathcal{A})$). Further, both the inclusions may be strict as shown by the following example.

Example 18. Consider the TMPA $\widetilde{\mathcal{A}}$ and TMSC $\widetilde{T}$ from Figure 3.4. All states of the TMPA are assumed to be initial and final and we have abstracted away the message contents for simplicity. Then we can observe that there is a timed linearization of $\widetilde{T}$ which is in $\mathcal{L}_{tw}(\widetilde{\mathcal{A}})$, namely, $(p!q, 5)(r!s, 5)(q?p, 6)(s?r, 7)$. But the timed word $(r!s, 5)(p!q, 5)(q?p, 6)(s?r, 7)$ is a timed linearization of $\widetilde{T}$ which is not in $\mathcal{L}_{tw}(\widetilde{\mathcal{A}})$. Thus, $\widetilde{T} \in \mathcal{L}_{time}(\widetilde{\mathcal{A}})$ but $\widetilde{T} \notin \mathcal{L}^{\forall}_{time}(\widetilde{\mathcal{A}})$ and we have t-lin($\mathcal{L}^{\forall}_{time}(\widetilde{\mathcal{A}})$) $\subsetneq \mathcal{L}_{tw}(\widetilde{\mathcal{A}}) \subsetneq$ t-lin($\mathcal{L}_{time}(\widetilde{\mathcal{A}})$).

Indeed when we restrict to TMPA that are locally timed, these three notions coincide and we have a perfect correspondance between the set of timed linearizations accepted and the set of TMSCs recognized. Thus, for a locally timed TMPA $\mathcal{A}$, we have t-lin($\mathcal{L}_{time}(\mathcal{A})$) = $\mathcal{L}_{tw}(\mathcal{A})$.

We could now ask if there are less rigid restrictions having this same property. For instance, we could restrict only the resets to be local and allow the guards to mention clocks from other processes. In this case, if in a guard, we allow only those clocks that are related by the partial order (i.e., the corresponding events are not concurrent), then we could reach the same result. We will investigate such a restriction later in the chapter where we replace clocks by event clocks.

Now, we define a TMPA to be *B-bounded* if its underlying untimed MPA is B-bounded and it is said to be *bounded* if there exists such a B.

Lemma 3.7. *For a bounded TMPA $\mathcal{A}$, $\mathcal{L}_{tw}(\mathcal{A})$ is a timed regular language.*

Proof. If the untimed MPA is B-bounded, then by definition every reachable state $(\overline{s}, \chi)$ of $\mathcal{B}_{\mathcal{A}}$ is *B-bounded*, i.e, $|\chi(c)| \leq B$ for some $B \in \mathbb{N}_{>0}$. This implies that $\mathcal{B}_{\mathcal{A}}$ is a finite timed automaton and so $\mathcal{L}_{tw}(\mathcal{B}_{\mathcal{A}}) = \mathcal{L}_{tw}(\mathcal{A})$ is a timed regular language. □

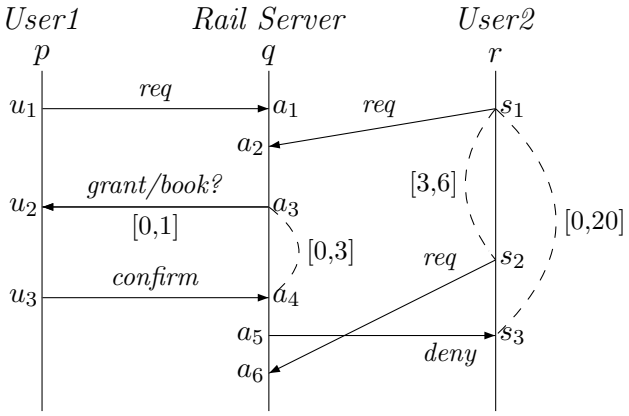


Figure 3.5: A TCMSC $\mathfrak{M}_1$ describing interaction of two users with a server

3.3 Message sequence charts with timing constraints

In the previous section, it may be noted that though TMPAs are distributed systems, for defining runs, we consider them operationally as global systems. This is not a coincidence, for, timed message sequence charts (and indeed timed linearizations) essentially capture only the operational/global behaviour of distributed systems.

In some sense, this is due to the fact that TMSCs and their timed linearizations are not very different! What we mean is that, by attaching time-stamping to events of an MSC, we lose much of its partial order information. The only partial-order information retained is through events on different processes with the same time-stamps.The remaining are totally ordered due to the global time-stamping.

A richer partial order behaviour can be retained by attaching *timing constraints* to pairs of events of the MSC. This approach has two other major advantages:

- Firstly, from a specification point of view, it allows the specifier to decide and enforce constraints between occurences of events as he chooses.
- Secondly, a single MSC with timing constraints can describe a whole family of TMSCs (with the same underlying MSC) thus being a much more succinct description of the timed behaviours of a system.

Example 19. Consider the same example as before, namely Example 15, where two users interact with a railway booking server. It might be that after granting a *User* request, the *Server* waits only for a bounded amount of time for him/her to respond before cancelling the request. The family of TMSCs satisfying such a constraint is easier to capture using a scenario with timing constraints as shown in

Figure 3.5. The label $[0, 3]$ from a_3 to a_4 specifies that *User1* must respond to the grant within 3 time units. The label $[0, 1]$ on the message from a_3 to u_3 specifies the bounds on the delay of message delivery and so on.

The TCMSC that we formally define below uses intervals from a set $\mathcal{S} \subseteq \mathcal{I}$ as timing constraints. Recall that $\mathcal{I}$ denotes the set of all intervals over the real line.

Definition 3.8. *Let $\mathcal{S} \subseteq \mathcal{I}$ be a set of intervals over the real line. An* MSC with timing constraints *or a* time-constrained MSC *(denoted* TCMSC*) over $(Act, \mathcal{S})$ is a pair $\mathfrak{M} = (M, \tau)$ where $M = (E, \leq, \lambda)$ is an MSC over Act and τ is a partial map from the set of irreflexive pairs of events, $(E \times E) \backslash \{(e, e) \mid e \in E\}$ to the set of intervals $\mathcal{S}$.*

As τ is a partial map, the domain of τ, denoted $\mathrm{dom}(\tau)$, consists of the pairs of events on which timing constraints are *imposed* by the TCMSC. When $\mathcal{S} = \mathcal{I}$, i.e, if $\mathfrak{M}$ is a TCMSC over $(Act, \mathcal{I})$, we ignore the latter component and say that $\mathfrak{M}$ is a TCMSC over Act. With this notation, it is clear that for any $\mathcal{S} \subseteq \mathcal{I}$, if $\mathfrak{M}$ is a TCMSC over $(Act, \mathcal{S})$, then $\mathfrak{M}$ is also a TCMSC over Act. As usual, when the set of actions is clear from context, we may ignore Act as well.

With this definition, TCMSCs can be considered as abstractions of TMSCs and timed words. Here and for the rest of this chapter, we let $\mathcal{S} \subseteq \mathcal{I}$ to be some fixed set of intervals.

Definition 3.9. *Let $\mathfrak{M} = (M, \tau)$ be a TCMSC over $(Act, \mathcal{S})$ with $M = (E, \leq, \lambda)$. A TMSC $T = (M, t)$ is said to* realize *$\mathfrak{M}$ if for all $(e_1, e_2) \in \mathrm{dom}(\tau)$ we have $|t(e_2) - t(e_1)| \in \tau(e_1, e_2)$. The set of all TMSCs that realize $\mathfrak{M}$ is denoted $\mathcal{L}_{time}(\mathfrak{M})$.*

Example 20. For instance, the TMSC of Figure 3.1 realizes the TCMSC from Figure 3.5 since each interval constraint between events in the TCMSC is satisfied by the time-stamps of the events in the TMSC.

This notion of MSCs with interval constraints on arbitrary pairs of events is in fact similar to the approach adopted by Alur et al. [7]. Thus we can use their MSC analysis tool to check consistency of these timing constraints in an MSC.

In the above definition, we can a priori define timing constraints between ANY two distinct but arbitrary events. But is this really what we want? In fact, this might defeat our purpose for introducing timing constraints in MSCs in the first place. For instance, in the TCMSC in Figure 3.5, should a specifier be allowed to have a *choice* of imposing constraints between the first event of User1 and the first event of User2? Or, from an implementation point of view, can a machine that implements such a constraint really be called a distributed machine?

In other words, the vital question is how flexible we want this timing to be, i.e, between which pairs of events we allow constraints. To define this formally, we fix an MSC $M = (E, \leq, \lambda)$ over Act and define the following relations that will relate the pairs of events on which we wish to impose timing constraints. First,

we denote the set of all irreflexive pairs of E by $\mathrm{P}(E) = \{(e, e') \in E \times E \mid e \neq e'\}$. Then,

- The *local process relation* which is defined as:
  $\mathrm{Loc}^M = \{(e, e') \in \mathrm{P}(E) \mid e <_{pp} e' \text{ for some } p \in Proc\}$
- The *message relation*, whose definition we recall:
  $\mathrm{Msg}^M = \{(e, e') \in \mathrm{P}(E) \mid e <_{pq} e' \text{ for some } (p, q) \in Ch\}$
- The *previous occurence of an action* $a \in Act$ is defined as:
  $\mathrm{Prev}_a^M = \{(e, e') \in \mathrm{P}(E) \mid \lambda(e') = a, e' \leq e,\ (e'' \leq e \wedge \lambda(e'') = a) \implies e'' \leq e'\}$
- The *next occurence of an action* $a \in Act$ is defined as:
  $\mathrm{Next}_a^M = \{(e, e') \in \mathrm{P}(E) \mid \lambda(e') = a, e \leq e',\ (e \leq e'' \wedge \lambda(e'') = a) \implies e' \leq e''\}$

Now we examine some approaches for defining subclasses of TCMSCs which are natural from the point of view of a specifier. Also, in the next sections and chapters, we will explicitly provide implementations and specifications for these restricted TCMSCs as well investigate other interesting properties. The first subclass of MSCs with timing constraints that we introduce, restricts timing to events on the same process and across messages. In other words,

Definition 3.10. *An* MSC with local timing constraints *over* $(Act, \mathcal{S})$*, denoted* loc-TCMSC *is a TCMSC* $\mathfrak{M} = (M, \tau)$ *over* $(Act, \mathcal{S})$ *such that* $\mathrm{dom}(\tau) \subseteq (\mathrm{Msg}^M \cup \mathrm{Loc}^M)$.

Example 21. The TCMSC $\mathfrak{M}_1$ from Figure 3.5 is an example of an MSC with local timing constraints. Indeed, we can observe that all timing constraints occuring in $\mathfrak{M}_1$ are either local to a process or across messages.

A more flexible timing is to allow timing between the next and previous occurence of any action from an event in the MSC along with the messages.

Definition 3.11. *An* MSC with ec-timing constraints *over* $(Act, \mathcal{S})$*, which we denote* ecTCMSC*, is a TCMSC* $\mathfrak{M} = (M, \tau)$ *over* $(Act, \mathcal{S})$ *such that* $\mathrm{dom}(\tau) \subseteq (\mathrm{Msg}^M \cup (\bigcup_{a \in Act} (\mathrm{Next}_a^M \cup \mathrm{Prev}_a^M)))$.

Example 22. Consider the ecTCMSC $\mathfrak{M}_2$ from Figure 3.6. We have abstracted away the message contents $\mathcal{M}$ for simplicity. $\mathfrak{M}_2$ has timing constraints defined between the following pairs of events: The first pair (w_3, v_3) is a message constraint. The second pair (w_1, u_1) is related by $\mathrm{Prev}_{p!q}$. However, note that this constraint can also be seen as between (u_1, w_1) which are related by the $\mathrm{Next}_{r?p}$. In such cases, our definition requires us to have the same interval as a constraint.

Finally, the third pair of events (u_3, v_4) are related by the $\mathrm{Next}_{q?p}$ relation. Here, it is interesting to observe that this pair of events are not related by *Msg* or

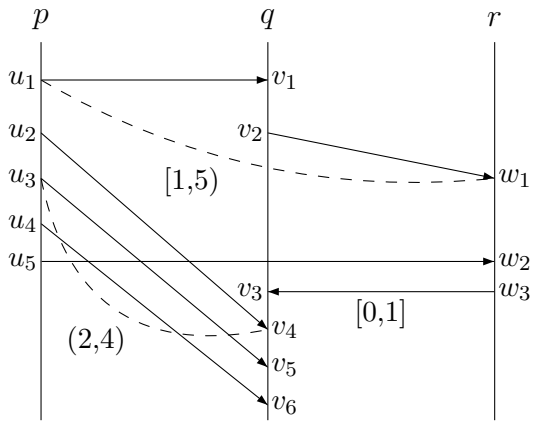


Figure 3.6: An ecTCMSC $\mathfrak{M}_2$

Prev_a relation for any $a \in Act$. Similarly, the pair of events (u_3, v_5) can be timed only by a message constraint. And again timing is allowed between the pair (v_5, u_4) only because of the $\text{Prev}_{p!q}$ relation. This illustrates to us that the expressivity of the relations Prev_a, *Msg* and Next_a are incomparable. Thus, we cannot subsume one by the other.

Now, if all possible allowed pairs have explicit timing constraints defined, then we call such a ecTCMSC to be *maximally defined*.

Definition 3.12. *An ecTCMSC* $\mathfrak{M} = (M, \tau)$ *over* $(Act, \mathcal{S})$ *is said to be* maximally defined *if* $\text{dom}(\tau) = (\text{Msg}^M \cup (\bigcup_{a \in Act} (\text{Next}_a^M \cup \text{Prev}_a^M)))$.

3.4 Event clock message passing automata

We have tried to motivate that the TCMSCs introduced in the previous section are the ideal formalism for describing timed and distributed scenarios or behaviours. In this section we introduce a timed system which is an implementation model for ecTCMSCs. In fact, we have chosen our timing constraints wisely so that the ECMPAs that we introduce below are the timed extensions of ECA introduced in the untimed case. Thus ECMPAs can be seen as event-clock versions of TMPAs just as ECAs were event-clock versions of timed automata. However, ECMPAs are in general be incomparable to locally timed TMPA, since we allow clocks across processes in the former, while the latter allows timing between any two events within a process.

First, we fix a formal set of symbols TC as follows:

$$\text{TC} = \{\text{Msg}\} \cup \{\text{Prev}_a \mid a \in Act\} \cup \{\text{Next}_a \mid a \in Act\} \tag{3.1}$$

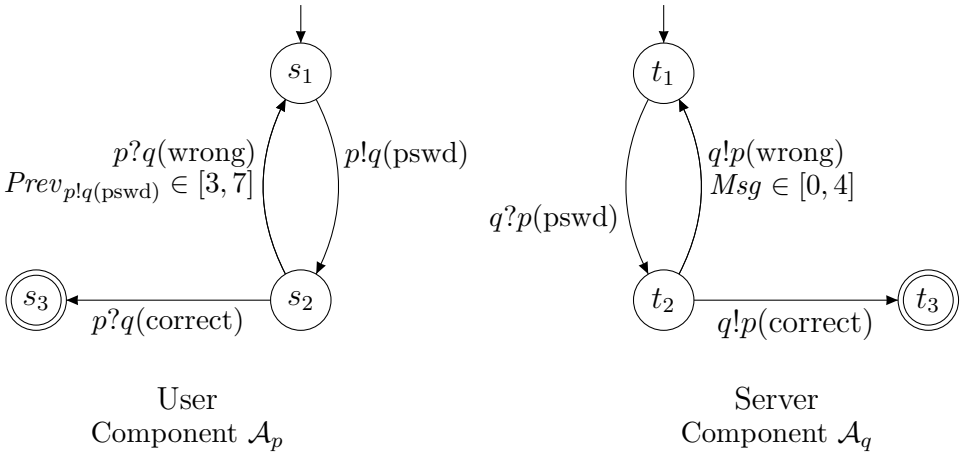


Figure 3.7: An ECMPA $\mathcal{A}_1$

When interpreted over an ecTCMSC $\mathfrak{M} = (M, \tau)$ over $(Act, \mathcal{S})$, each symbol $\alpha \in TC$ is interpreted as the relation α^M defined in the previous section. We let $\text{TC}^M = \bigcup_{\alpha \in \text{TC}}(\alpha^M)$. Also, $[\text{TC} \dashrightarrow \mathcal{I}]$ denotes the set of partial maps from the set of symbols TC to the set of intervals $\mathcal{I}$.

Definition 3.13. *An* event clock message passing automaton *(ECMPA) is a tuple* $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, Act, \Delta, F)$*, where*

- Δ *is a finite set of auxiliary messages,*
- Act *is the alphabet*
- *For each* $p \in Proc$*, the component* $\mathcal{A}_p$ *is a structure* $(S_p, \rightarrow_p, \iota_p)$ *where*
  - S_p *is a finite set of p-local states*
  - $\iota_p \in S_p$ *is the p-local initial state*
  - $\rightarrow_p$ *is a* finite *subset of* $(S_p \times Act_p \times [\text{TC} \dashrightarrow \mathcal{I}] \times \Delta \times S_p)$
- $F \subseteq \prod_{p \in Proc} S_p$ *is a set of global final states.*

Intuitively the automaton model is just an MPA extended with guards using clocks from TC. That is, each guard g is a partial map of the form $g \in [\text{TC} \dashrightarrow \mathcal{I}]$.

Example 23. A simple example of an event clock message passing automaton is illustrated in Figure 3.7. The ECMPA $\mathcal{A}_1$ describes the interaction between a user and a server, with the server trying to authenticate the user. The user components is denoted $\mathcal{A}_p$ and the server $\mathcal{A}_q$ as shown. The set of actions Act consists

of: $p!q(pswd)$ (user sends password to server), $q?p(pswd)$ (server receives password), $q!p(correct), q!p(wrong)$ (server sends appropriate message to user) and $p?q(correct), p?q(wrong)$ (user receives message from server). Thus, in the above automaton, the user starts by sending its password. If the password received by server is correct, it acknowledges this and goes to final state. Else, it sends message *wrong* which must reach in 4 time units and waits in state t_1. If the user receives *correct* it goes to its final state. If not, it must receive *wrong* in a bounded amount of time since it last sent its password. And in this case, the user tries to resend password. Otherwise, the current interaction is considered void and so the run is rejected.

3.4.1 Semantics over TCMSCs

Formally, we define the run of an ECMPA $\mathcal{A}$ over an ecTCMSC $\mathfrak{M} = (M, \tau)$ over $(Act, \mathcal{S})$, where $M = (E, \leq, \lambda)$. Again consider $r : E \to \bigcup_{p \in Proc} S_p$ labeling each event of process p with a local state from S_p. Define $r^- : E \to \bigcup_{p \in Proc} S_p$ as before: For event $e \in E_p$, if there is another event $e' \in E_p$ such that $e' \lessdot_{pp} e$, then $r^-(e) = r(e')$ and $r^-(e) = \iota_p$ otherwise. Then r is a *run* of $\mathcal{A}$ on $\mathfrak{M}$ if, for all $e, e' \in E$, such that $e <_{pq} e'$ for some channel $(p, q) \in Ch$, there are $g, g' \in [\mathrm{TC} \dashrightarrow \mathcal{I}]$ and a control message $d \in \Delta$ such that,

- $(r^-(e), \lambda(e), g, d, r(e)) \in \to_p$ and $(r^-(e'), \lambda(e'), g', d, r(e')) \in \to_q$, (3.2)
- $\forall \alpha \in \mathrm{dom}(g), \exists \widetilde{e} \in E$ s.t., $(e, \widetilde{e}) \in \alpha^M$ and $\tau(e, \widetilde{e}) \subseteq g(\alpha)$, (3.3)
- $\forall \alpha \in \mathrm{dom}(g'), \exists \widetilde{e'} \in E$ s.t., $(e', \widetilde{e'}) \in \alpha^M$ and $\tau(e', \widetilde{e'}) \subseteq g'(\alpha)$. (3.4)

Note that given e, e' as above and α^M, $\widetilde{e}$ and $\widetilde{e'}$ are uniquely defined since α^M is, in fact, a partial function. We define $s_p = r(e_p)$, where e_p is the maximal event in the p^{th} process. If there are no events on process p, we set $s_p = \iota_p$. Then run r is *successful* if $(s_p)_{p \in Proc} \in F$. A TCMSC over $(Act, \mathcal{S})$ is *accepted* by an ECMPA $\mathcal{A}$ if it admits a successful run. We denote by $\mathcal{L}_{TC}(\mathcal{A})$, the set of all TCMSCs over Act that are accepted by $\mathcal{A}$. We remark that, since a TCMSC over $(Act, \mathcal{S})$ for $\mathcal{S} \subseteq \mathcal{I}$ is also a TCMSC over Act, $\mathcal{L}_{TC}(\mathcal{A})$ is the set of all TCMSCs over $(Act, \mathcal{S})$ for all interval sets $\mathcal{S} \subseteq \mathcal{I}$.

Example 24. As an example, we can observe that the ecTCMSC $\mathfrak{M}_3$ from Figure 3.8 is accepted by $\mathcal{A}_1$, the ECMPA shown in Figure 3.7. To see this, first note that $\mathfrak{M}_3 = (M, \tau)$ where $M = (E, \leq, \lambda)$ with $E = \{u_i, v_i \mid i \in \{1, \ldots, 4\}\}$, $\mathrm{dom}(\tau) = \{(v_2, u_2), (u_1, u_2)\}$ and $\tau(v_2, u_2) = [0, 3], \tau(u_1, u_2) = [3, 7]$. Also, the set of local states of $\mathcal{A}_1$ is $S = \{s_i, t_i \mid i \in \{1, \ldots, 3\}\}$.

Now, consider the map $r : E \to S$ such that $r(u_1) = r(u_3) = s_1$, $r(v_1) = r(v_3) = t_2$, $r(u_2) = s_2$,$r(v_2) = t_2$ and finally $r(u_4) = s_3$ and $r(v_4) = t_3$. Then, it is easy to see that Condition (3.2–3.4) hold. Further, (s_3, t_3) is a final state and so r is an accepting run of $\mathcal{A}_1$ on $\mathfrak{M}_3$.

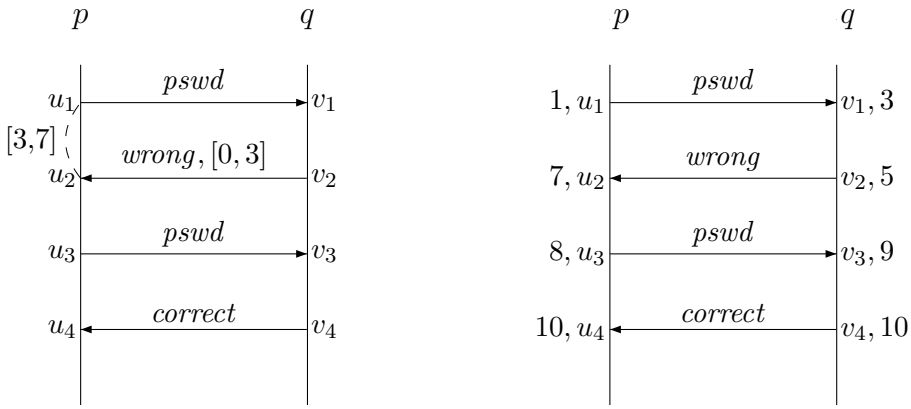


Figure 3.8: ecTCMSC $\mathfrak{M}_3$ and TMSC T_3

3.4.2 Semantics over TMSCs

Since ecTCMSCs are also abstractions of TMSCs, we get the semantics of ECMPAs over TMSCs. Unsurprisingly, this can also be done directly by defining runs of ECMPAs on TMSCs rather than TCMSCs. The definition of a run of $\mathcal{A}$ over a TMSC $T = (M, t)$ is the same as over a TCMSC $\mathfrak{M} = (M, \tau)$, except that conditions (3.3) and (3.4) are respectively replaced by (3.5) and (3.6) defined below:

- for all $\alpha \in \mathrm{dom}(g), \exists \widetilde{e} \in E$ s.t., $(e, \widetilde{e}) \in \alpha^M$ and $|t(\widetilde{e}) - t(e)| \in g(\alpha)$ (3.5)
- for all $\alpha \in \mathrm{dom}(g'), \exists \widetilde{e}' \in E$ s.t., $(e', \widetilde{e}') \in \alpha^M$ and $|t(\widetilde{e}') - t(e')| \in g'(\alpha)$ (3.6)

Then, with the same notion of acceptance as above, we can denote the set of all TMSCs accepted by a given ECMPA $\mathcal{A}$ as $\mathcal{L}_{time}(\mathcal{A})$.

Example 25. Again, the TCMSC T_3 from Figure 3.8 is accepted by the ECMPA Aut_1 shown in Figure 3.7. The set of events of T_3 are the same as the set of events of $\mathfrak{M}_3$ and the map described in Example 24 can easily be seen to be an accepting run of $\mathcal{A}_1$ on T_3 as well. We may also notice here that T_3 realizes $\mathfrak{M}_3$. This is not a coincidence. In fact, we can make the following general observation,

Lemma 3.14. *Suppose a TMSC T over Act realizes a TCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$. Then for an ECMPA $\mathcal{A}$, $\mathfrak{M} \in \mathcal{L}_{TC}(\mathcal{A})$ implies $T \in \mathcal{L}_{time}(\mathcal{A})$.*

Proof. The lemma essentially follows from the definitions given above. Let the $T = (M, t)$ be the TMSC over Act with $M = (E, \leq, \lambda)$ and $\mathfrak{M} = (M, \tau)$. Also let $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, Act, \Delta, F)$ as in Definition 3.13. Then, any run r of $\mathcal{A}$ on $\mathfrak{M}$ satisfies the conditions (3.2–3.4).

Now, since T realizes $\mathfrak{M}$, for all $(e_1, e_2) \in \mathrm{dom}(\tau)$, we have $|t(e_1) - t(e_2)| \in \tau(e_1, e_2)$. This along with (3.3), implies that for all $\alpha \in \mathrm{dom}(g)$ there exists $\widetilde{e} \in E$ such that $(e, \widetilde{e}) \in \alpha^M$ and $|t(e) - t(\widetilde{e})| \in \tau(e, \widetilde{e}) \subseteq g(\alpha)$. Thus, we find that (3.5)

holds. Similarly, from (3.4) we obtain (3.6). Hence we can conclude that r is a run of $\mathcal{A}$ on T.

Since every run of $\mathcal{A}$ on $\mathfrak{M}$ is also a run of $\mathcal{A}$ on T and the acceptance criterion is the same in both cases we conclude that if $\mathfrak{M} \in \mathcal{L}_{TC}(\mathcal{A})$ then $T \in \mathcal{L}_{time}(\mathcal{A})$. □

The converse however, holds only under some additional assumptions on $\mathfrak{M}$ which will be seen in the next chapter.

3.5 Time constrained message sequence graphs

Just as we used MSGs to describe infinite families of MSCs, to describe infinite families of TCMSCs, we label the nodes of an MSG with TCMSCs instead of normal MSCs. We also permit process-wise timing constraints along the edges of the MSG. A constraint for process p along an edge $q \to q'$ specifies a constraint between the final p-event of $\Phi(q)$ and the initial p-event of $\Phi(q')$, provided p actively participates in both these nodes. If p does not participate in either of these nodes, the constraint is ignored.

Definition 3.15. *A* time-constrained message sequence graph (TCMSG)*, denoted* $\mathfrak{G}$*, is a tuple* $(G, \mathcal{L}^{TC}, \Phi, EdgeC)$ *where*

- G *is a graph* $(V, \to, v_{in}, V_F)$*,*
- $\mathcal{L}^{TC}$ *is a finite set of non-empty TCMSCs over Act whose sets of events are disjoint.*
- $\Phi : V \to \mathcal{L}^{TC}$ *is a map assigning a TCMSC to every vertex of the graph.*
- $EdgeC : ((\to) \times Proc) \to \mathcal{I}$ *is map which assigns to each edge of the graph a time interval per process.*

In the following, we assume that there is a default interval $(-\infty, \infty)$ attached, whenever the constraint is not explicitly defined.

Semantics over TCMSCs

For a given TCMSG $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$, for each vertex v of G, let $\Phi(v) \in \mathcal{L}^{TC}$ be the TCMSC $\mathfrak{M}_v = (M_v, \tau_v)$ where $M_v = (E^v, \leq^v, \lambda_v)$. Then we denote the set of all events by $E^{\mathfrak{G}} = \bigcup_{v \in V} E^v$.

Now, for a non-empty path π in G, we define the *TCMSC generated by* π to be $\mathfrak{M}_\pi = (M_\pi, \tau_\pi)$ where,

- we recall that $M_\pi = (E^\pi, \leq^\pi, \lambda_\pi)$ is such that

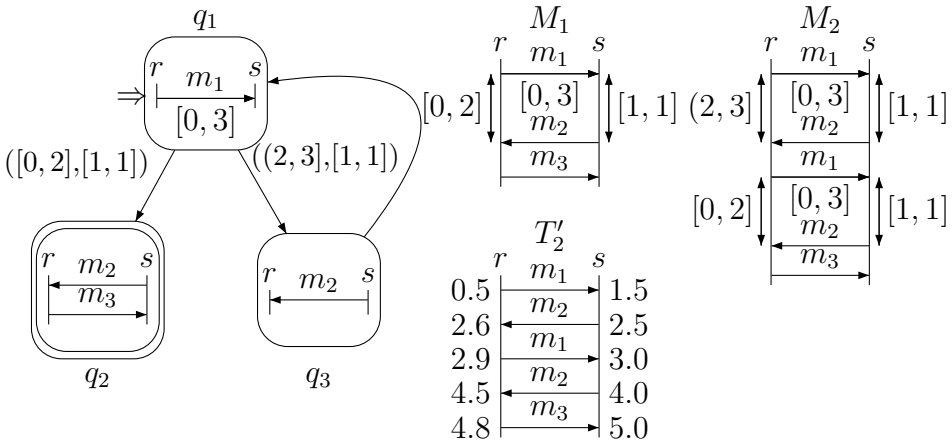


Figure 3.9: A TCMSG and some TCMSCs that it generates

- $E^\pi = \bigcup_{\rho v \preceq \pi} E^v \times \{\rho v\}$
- 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 $e, e' \in E_p$ and either $\rho v \preceq \rho' v'$ or $\rho v = \rho' v'$ and in this case $e <^v_{pp} e'$.
  - $(e, \rho v) <^\pi_{pq} (e', \rho' v')$ for some processes $p \neq q$, if $\rho v = \rho' v'$ and $e <^v_{pq} e'$.

- and $\tau_\pi \in [(E^\pi \times E^\pi) \dashrightarrow \mathcal{I}]$ is a partial map such that

$$\tau_\pi((e, \rho v), (e', \rho' v')) = \begin{cases} \tau_v(e, e') & \text{if } \rho v = \rho' v' \\ EdgeC((v, v'), p) & \text{if } \rho' = \rho v, \text{ and} \\ & e = max(E^v_p), e' = min(E^{v'}_p) \end{cases}$$

The language $\mathcal{L}_{TC}(\mathfrak{G})$ is defined to be the set of all TCMSCs over Act generated by paths in G that start from the initial vertex and end in a final vertex.

Example 26. Figure 3.9 shows a TCMSG and some of the TCMSCs that it generates. We omit the trivial constraints (which set the default value of true to each unspecified edge) to avoid cluttering the figure.

In this chapter we have essentially introduced all our models to describe as well as implement timed and distributed scenarios. We have also tried to motivate the inherent differences and conflicts that we need to resolve. In the next three chapters, we will both ask and solve questions about the formalisms introduced here.

4

Logical Characterization of ECMPAs

One of the cornerstone results of formal language theory is the connection between automata theory and classical logic. This was established over words by Büchi and Elgot early in the sixties [20, 37], who showed that the language of a finite automaton can be expressed by a formula from monadic second-order logic (MSO) and that, conversely, any MSO formula can be effectively transformed into an equivalent finite automaton. The study of relations between logical formalisms and operational automata has had many attempts at generalization including trying to extend and abstract the definition of words themselves.

As mentioned in Chapter 2, this equivalence was lifted to the partial order setting (Section 2.2.5) and the timed setting (Section 2.3.4). Our goal in this chapter is to merge these two approaches by looking at partial orders with timing information. For this, we use the TMSCs which we introduced in the previous chapter as a formalism to describe the executions of timed and partial systems (the TMPA and the ECMPA).

We prove the equivalence between the logic and ECMPA over TMSCs with and without existential bounds on channels. However, to do this, we actually use the TCMSCs as a finite description of an infinite collection of TMSCs thus again showing the significance of the TCMSCs.

We begin by introducing the logical framework for timed partial orders.

4.1 Timed monadic second-order logic

As usual, we start with a supply of individual variables $x, y, \ldots$, and set variables $X, Y, \ldots$ which range over events (and sets of events) of the timed MSC. We generalize the timed logic introduced in Section 2.3.4 by using (other than unary predicates $P_a(x)$ for $a \in Act$) generalized timing predicates of the form $\delta_\alpha(x) \in I$ for a variable x, $\alpha \in \mathrm{TC}$ and $I \in \mathcal{I}$. Here TC is the same set of symbols defined by Equation (3.1) of the previous chapter and are interpreted as relations over the events. Again, as for MSO over MSCs introduced in Section 2.2.4, the logic depends on a set $\mathfrak{R}$ of (binary) relation symbols, which settles the access to the

partial order relation. Thus,

Definition 4.1. *The set* TMSO$(Act, \mathfrak{R})$ *of all timed monadic second-order logic formulae over Act with the relational symbols from* $\mathfrak{R}$*, is generated inductively using the following grammar:*

$$\varphi \ ::= \ P_a(x) \mid x \in X \mid x = y \mid R(x,y) \mid \delta_\alpha(x) \in I \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi$$

where, x, y *are individual variables,* X *is a set variable,* $a \in Act, R \in \mathfrak{R}, \alpha \in$ TC *and* $I \in \mathcal{I}$*.*

Again, the *existential* fragment of TMSO$(Act, \mathfrak{R})$, denoted ETMSO$(Act, \mathfrak{R})$, comprises of all formulas $\exists X_1 \ldots \exists X_n \varphi$ such that φ does not contain any set quantifier.

Though we have defined the logic above for arbitrary sets of relational symbols $\mathfrak{R}$, we are in fact interested only in the two restricted sets that we have seen before, namely $\mathfrak{R}_{\leq}$ and $\mathfrak{R}_{<}$. Recall, from Section 2.2.4 that $\mathfrak{R}_{\leq} = \{\leq\}$ and $\mathfrak{R}_{<} = \{<_{pp} \mid p \in Proc\} \cup \{<_{pq} \mid p \neq q\}$. Then a formula φ from any of these logics, i.e, TMSO$(Act, \mathfrak{R}_{\leq})$, TMSO$(Act, \mathfrak{R}_{<})$, ETMSO$(Act, \mathfrak{R}_{\leq})$ or ETMSO$(Act, \mathfrak{R}_{<})$ can be interpreted over TMSCs as well as TCMSCs as follows. We write TMSO(Act) or TMSO to denote the union of all formulae from these logics, when there is no scope for confusion.

4.1.1 Semantics over TMSCs

We start by observing that, like a timed word, a TMSC $T = (M, t)$ over Act for $M = (E, \leq, \lambda)$ is an $\mathbb{R}_{\geq 0}$-extended labeled relational structure $\mathfrak{A} = (E, \mathfrak{R}^{\mathfrak{A}}, \lambda, \rho)$ over Act with $\mathfrak{R}^{\mathfrak{A}} = \{\leq^{\mathfrak{A}}\}$, $\rho = t$. Therefore it is not surprising that the definition of the semantics of a TMSO formula over TMSCs is identical to that of ecMSO over timed words from Section 2.3.4 as we see below.

Thus, for the TMSC T, let $\mathbb{I}$ be an *interpretation* mapping first order variables to elements in E and second order variables to subsets of E. Then, for $\varphi \in$ TMSO(Act), we define the satisfaction relation $T, \mathbb{I} \models \varphi$, by induction on the structure of φ. For all operators, except the timing predicate, this is given by Conditions (2.1–2.8) from Section 2.1.2. For instance, by Condition (2.1), the unary predicate $P_a(x)$ expresses that $\mathbb{I}(x)$ is labeled with $a \in Act$, i.e., $\lambda(\mathbb{I}(x)) = a$.

The only novelty is the timing predicate, for which we define the satisfaction relation as follows. Intuitively, by $\delta_\alpha(x) \in I$ we mean that there is an event $e \in E$ such that $\mathbb{I}(x)$ and e are related by α^M and the time difference between these events $\mathbb{I}(x)$ and e is contained in I. Formally for each $\alpha \in$ TC we define,

$$T, \mathbb{I} \models \delta_\alpha(x) \in I \text{ if } \exists e \in E, \text{ s.t., } (\mathbb{I}(x), e) \in \alpha^M \text{ and } |t(e) - t(\mathbb{I}(x))| \in I \qquad (4.1)$$

Then, as usual, for sentences φ (i.e, formulae without free variables) we write $T \models \varphi$ instead of $T, \mathbb{I} \models \varphi$ and denote by $\mathcal{L}_{time}(\varphi)$ the set of all TMSCs T over Act such $T \models \varphi$.

4.1.2 Semantics over TCMSCs

Now, turning to ecTCMSCs, we observe that an ecTCMSC $\mathfrak{M} = (M, \tau)$ over $(Act, \mathcal{S})$ with $M = (E, \leq, \lambda)$ is, in fact, a $\mathcal{S}$-extended labeled relational structure $(E, \mathfrak{R}^{\mathfrak{A}}, \lambda, \rho)$ over Act where $\mathfrak{R}^{\mathfrak{A}} = \{\leq^{\mathfrak{A}}\}$, $\rho = \tau$ and $\mathcal{S} \subseteq \mathcal{I}$ is a set of intervals.

Thus, we can give a formula $\varphi \in \text{TMSO}(Act)$ a natural semantics over $\mathfrak{M}$ exactly as done for TMSCs above. The only noteworthy difference is in the timing predicate $\delta_\alpha(x) \in I$, where for any $\alpha \in \text{TC}$, we define

$$\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I \text{ if } \exists e \in E \text{ s.t., } (\mathbb{I}(x), e) \in \alpha^M \text{ and } \tau(\mathbb{I}(x), e) \subseteq I \tag{4.2}$$

The set of ecTCMSCs over Act that satisfy a TMSO sentence φ is denoted by $\mathcal{L}_{TC}(\varphi)$.

Example 27. Let us again consider the interaction scenario modeled in Example 23, where a server tries to authenticate a user. Thus the set of actions Act is the same as in Example 23. Then, suppose we wish to specify that every Message *wrong* sent by the server takes no more than 4 units of time to be conveyed. This can be written as the following sentence in our logic:

$$\forall x \; P_{q!p(wrong)}(x) \rightarrow \delta_{Msg}(x) \in [0, 4] \tag{4.3}$$

Similarly, we may require that whenever Message *wrong* is received by the user, the time elapsed since it last sent its password must be at least 3 units and at most 7 units. This can be easily expressed as:

$$\forall x \; P_{p?q(wrong)}(x) \rightarrow \delta_{\text{Prev}_{p!q(pswd)}}(x) \in [3, 7] \tag{4.4}$$

We observe immediately that both of these sentences (4.3) and (4.4) are satisfied by ecTCMSC $\mathfrak{M}_3$ and TMSC T_3 from Figure 3.8. Then, we have the following lemma for the logic which corresponds to Lemma 3.14 for ECMPA.

Lemma 4.2. *Let φ be a TMSO formula and let T be a TMSC over Act which realizes an ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$ for some $\mathcal{S} \subseteq \mathcal{I}$. Then, $\mathfrak{M} \in \mathcal{L}_{TC}(\varphi)$ implies $T \in \mathcal{L}_{time}(\varphi)$.*

Proof. Let $T = (M, t)$ be the TMSC over Act with $M = (E, \leq, \lambda)$. And for $\mathcal{S} \subseteq \mathcal{I}$, $\mathfrak{M} = (M, \tau)$ is the ecTCMSC over $(Act, \mathcal{S})$ such that T realizes $\mathfrak{M}$. Then, we show the lemma for any interpretation $\mathbb{I}$, by induction on structure of φ. The only interesting case is the timing predicate. The others are routine deductions.

For this case, we show that if $\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I$ then $T, \mathbb{I} \models \delta_\alpha(x) \in I$. By definition, $\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I$ means that there exists $e \in E$, such that $(\mathbb{I}(x), e) \in \alpha^M$ and $\tau(\mathbb{I}(x), e) \subseteq I$. But T realizes $\mathfrak{M}$ and so we have $|t(e) - t(\mathbb{I}(x))| \in \tau(\mathbb{I}(x), e) \subseteq I$. Thus, by definition, we conclude that $T, \mathbb{I} \models \delta_\alpha(x) \in I$. □

The converse also holds but in a restricted case, as we will see next.

4.2 From TMSCs to TCMSCs

A TMSC that realizes a TCMSC can be thought of as its realization. Thus, if the TCMSC exhibits a property, the corresponding TMSC also does so. This is illustrated in Lemmas 3.14 and 4.2 where the *property* is having a run on an automaton, or satisfying a formula. In this section, we are interested in the converse question. In other words, if a TMSC exhibits a property, when can we say that a TCMSC that it realizes also exhibits the same property?

For this, given a TMSC, we derive a canonical representative ecTCMSC using intervals from a specific set which depends on the property. Then it turns out that, this representative exhibits the property if and only if a TMSC realizing it exhibits the property. We formalize these vague ideas in this section.

We begin by introducing the notion of a *proper interval set* from [34], which will play an important role in what follows.

Definition 4.3. *A set of intervals $\mathcal{S} \subseteq \mathcal{I}$ is said to be* proper *if it forms a finite partition of $\mathbb{R}^{\geq 0}$.*

Definition 4.4. *An interval set $\mathcal{S}$ is said to* refine *another interval set $\mathcal{S}'$ if every interval $I' \in \mathcal{S}'$ is the union of some collection of intervals of $\mathcal{S}$, i.e, $I' = \bigcup_{I \in \mathcal{S}} I$.*

Example 28. Consider the set of intervals $\mathcal{S}_1 = \{[0,4],[3,7]\}$. Then, we may observe that the interval set $\mathcal{S}_2 = \{[0,3),[3,4],(4,7]\}$ refines $\mathcal{S}_1$ and if we add the interval $(7,\infty)$ to $\mathcal{S}_2$ we obtain a proper interval set $\mathcal{S}_3$ that refines $\mathcal{S}_1$.

We can now make the following elementary observations,

Remark 4.5. If $\mathcal{S}$ is a proper interval set which refines another interval set $\mathcal{S}'$, then for all $I \in \mathcal{S}$ and $I' \in \mathcal{S}'$, we have, either $I \subseteq I'$ or $I \cap I' = \emptyset$.

Proposition 4.6. *For any finite interval set, there exists a proper interval set that refines it.*

Proof. Let $\mathcal{S} \subseteq \mathcal{I}$ be a finite interval set. Then, we define a canonical proper interval set of $\mathcal{S}$ denoted $\text{prop}(\mathcal{S})$ as follows: Let $R = (t_1, \ldots, t_n)$ be the sequence of bounds that appear in $\mathcal{S}$, arranged in increasing order $t_1 < \ldots < t_n$ and which are different from $0, \infty$. If R is empty, we define $\text{prop}(\mathcal{S}) = \{[0,\infty)\}$. Otherwise, we define $\text{prop}(\mathcal{S}) = \{[0,0],(0,t_1),[t_1,t_1],(t_1,t_2),\ldots,[t_n,t_n],(t_n,\infty)\}$.

With this definition it follows that $\text{prop}(\mathcal{S})$ is a proper interval set and that $\text{prop}(\mathcal{S})$ refines $\mathcal{S}$. □

Now we can show that for a proper interval set $\mathcal{S}$ and a TMSC T, there is a unique maximally defined ecTCMSC using intervals only from $\mathcal{S}$ such that T realizes it. Formally,

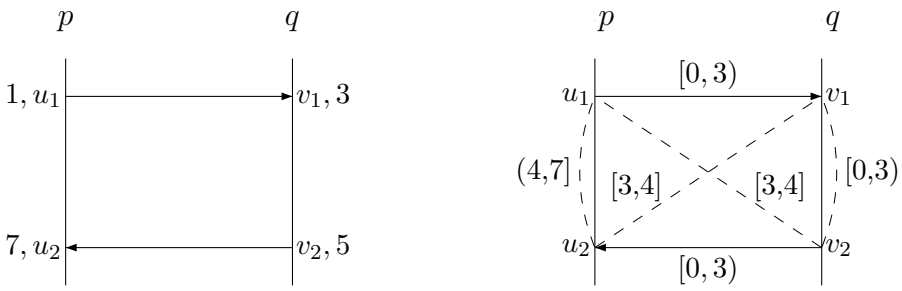


Figure 4.1: TMSC T_4 and its representative ecTCMSC $\mathfrak{M}_{T_4}^{\mathcal{S}_3}$

Lemma 4.7. *Let $\mathcal{S}$ be a proper interval set and $T = (M, t)$ be a TMSC over Act. Then, there exists a unique ecTCMSC $\mathfrak{M} = (M, \tau)$ over $(Act, \mathcal{S})$ such that $\tau : TC^M \to \mathcal{S}$, T realizes $\mathfrak{M}$ and $\mathfrak{M}$ is maximally defined.*

Proof. We first observe that, for each $(e, e') \in \mathrm{TC}^M$, the real number $|t(e') - t(e)|$ is in a unique interval of $\mathcal{S}$. Thus, consider the maximally defined TCMSC defined as: $\mathfrak{M}_T^{\mathcal{S}} = (M, \tau)$ where, for any $(e, e') \in \mathrm{TC}^M$, $\tau(e, e')$ is defined to be the unique interval of $\mathcal{S}$ containing $|t(e') - t(e)|$. Then, T realizes $\mathfrak{M}_T^{\mathcal{S}}$ by definition and the uniqueness follows since $\mathcal{S}$ is a proper interval set. □

It turns out that, this unique ecTCMSC is the "canonical representative" for a TMSC, that we were searching for in the beginning of this section. We fix its notation with the following definition,

Definition 4.8. *For a proper interval set $\mathcal{S}$ and TMSC T, we define $\mathfrak{M}_T^{\mathcal{S}}$ to be the unique maximally defined ecTCMSC over $(Act, \mathcal{S})$ such that T realizes it.*

Example 29. Consider the TMSC T_4 from Figure 4.1, which represents a part of the scenario of TMSC T_3 from Figure 3.8. We have also abstracted away the message contents. Now, let $\mathcal{S}_3$ be the proper interval set $\mathcal{S}_3$ defined in Example 28. Then, the unique maximally defined ecTCMSC over $(Act, \mathcal{S}_3)$ which is realized by T_4 is the ecTCMSC $\mathfrak{M}_{T_4}^{\mathcal{S}_3}$ shown in Figure 4.1. Indeed, it is easily seen that it satisfies the condition described in the proof of Lemma 4.7.

Now, given a TMSO formula φ, we let $\mathrm{Int}(\varphi)$ denote the finite set of intervals I for which φ has a sub-formula of the form $\delta_\alpha(x) \in I$. Now look at any proper interval set $\mathcal{S}$ that refines $\mathrm{Int}(\varphi)$. By Proposition 4.6, there exists at least one, namely $\mathrm{prop}(\mathrm{Int}(\varphi))$.

Lemma 4.9. *Given a T-MSC T, a TMSO formula φ, and a proper interval set $\mathcal{S}$ that refines $\mathrm{Int}(\varphi)$, we have $T \models \varphi$ if and only if $\mathfrak{M}_T^{\mathcal{S}} \models \varphi$.*

Proof. Let $T = (M, t)$ be a T-MSC with $M = (E, \leq, \lambda)$. Then, by Lemma 4.7, we have the TCMSC $\mathfrak{M}_T^{\mathcal{S}} = (M, \tau)$ such that, for all $\alpha \in \mathrm{TC}$, and all $(e, e') \in \alpha^M$,

$|t(e')-t(e)| \in \tau(\alpha)$. Since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, one direction of this lemma follows from Lemma 4.2, i.e, we have $\mathfrak{M}_T^{\mathcal{S}} \models \varphi$ implies $T \models \varphi$.

We now show the other direction, for any interpretation $\mathbb{I}$, by induction on structure of φ. The only interesting case is the timing predicate. The others are routine deductions. Thus we will now show that $T, \mathbb{I} \models \delta_\alpha(x) \in I$ implies $\mathfrak{M}_T^{\mathcal{S}}, \mathbb{I} \models \delta_\alpha(x) \in I$.

Assume $T, \mathbb{I} \models \delta_\alpha(x) \in I$. Then by definition, there exists $e \in E$, such that $(\mathbb{I}(x), e) \in \alpha^M$ and $|t(e)-t(\mathbb{I}(x))| \in I$. Then, as $\mathfrak{M}_T^{\mathcal{S}}$ is maximally defined, $\tau(\mathbb{I}(x), e)$ exists and $\tau(\mathbb{I}(x), e) \in \mathcal{S}$. But $\mathcal{S}$ refines $\mathrm{Int}(\varphi)$ and $I \in \mathrm{Int}(\varphi)$, so we must have either $\tau(\mathbb{I}(x), e) \subseteq I$ or $\tau(\mathbb{I}(x), e) \cap I = \emptyset$. Now, since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, we have $|t(e)-t(\mathbb{I}(x))| \in \tau(\mathbb{I}(x), e)$. Therefore we get, $|t(e)-t(\mathbb{I}(x))| \in \tau(\mathbb{I}(x), e) \cap I$ and so we conclude that $\tau(\mathbb{I}(x), e) \subseteq I$. Thus, there exists $e \in E$, such that $(\mathbb{I}(x), e) \in \alpha^M$ and $\tau(\mathbb{I}(x), e) \subseteq I$ which by definition implies that $\mathfrak{M}_T^{\mathcal{S}}, \mathbb{I} \models \delta_\alpha(x) \in I$. □

We can do the same for ECMPAs as well. That is, given an ECMPA $\mathcal{A}$, let $\mathrm{Int}(\mathcal{A})$ denote the set of finite set of intervals that occur in $\mathcal{A}$ as guards. Again we can consider a proper interval set $\mathcal{S}$ which refines $\mathrm{Int}(\mathcal{A})$ since by Proposition 4.6, there exists at least one. Then again we have,

Lemma 4.10. *Given a TMSC T, an ECMPA $\mathcal{A}$ and a proper interval set $\mathcal{S}$ that refines $\mathrm{Int}(\mathcal{A})$, we have $T \in \mathcal{L}_{time}(\mathcal{A})$ if and only if $\mathfrak{M}_T^{\mathcal{S}} \in \mathcal{L}_{TC}(\mathcal{A})$.*

Proof. The proof essentially uses the same arguments as in the previous lemma. To see this, let us fix $T = (M, t)$ to be the TMSC over Act with $M = (E, \leq, \lambda)$, and $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, Act, \Delta, F)$ with $\mathcal{A}_p = (S_p, \iota_p, \rightarrow_p)$. Now, since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, by Lemma 3.14, we immediately obtain one direction of the result, i.e, $\mathfrak{M}_T^{\mathcal{S}} \in \mathcal{L}_{TC}(\mathcal{A})$ implies $T \in \mathcal{L}_{time}(\mathcal{A})$.

For the reverse direction, assume $T \in \mathcal{L}_{time}(\mathcal{A})$. Then by definition there is a successful run r of $\mathcal{A}$ on T. Since r is a run, for all $e, e' \in E$ such that $e <_{pq} e'$ for $(p, q) \in Ch$, we find $g, g' \in [TC \dashrightarrow \mathcal{I}]$ and $d \in \Delta$ such that conditions (3.2),(3.5) and (3.6) hold. We show then that r is also a run of $\mathcal{A}$ on $\mathfrak{M}_T^{\mathcal{S}}$, for which it is enough to show that conditions (3.3) and (3.4) hold.

We start from (3.5), which says that for each $\alpha \in \mathrm{dom}(g)$, there is $\widetilde{e} \in E$ such that $(e, \widetilde{e}) \in \alpha^M$. Since $\mathfrak{M}_T^{\mathcal{S}}$ is maximally defined, $\tau(e, \widetilde{e})$ exists and as T realizes $\mathfrak{M}_T^{\mathcal{S}}$ we obtain $|t(\widetilde{e}) - t(e)| \in \tau(e, \widetilde{e})$. But again from (3.5), $|t(\widetilde{e}) - t(e)| \in g(\alpha)$. Thus, we find that $\tau(e, \widetilde{e}) \cap g(\alpha) \neq \emptyset$. Now, $\tau(e, \widetilde{e}) \in \mathcal{S}$, $g(\alpha) \in \mathrm{Int}(\mathcal{A})$ and we know that $\mathcal{S}$ is a proper interval set that refines $\mathrm{Int}(\mathcal{A})$. Thus, we conclude that $\tau(e, \widetilde{e}) \subseteq g(\alpha)$ which shows that (3.3) holds. Similarly, (3.4) can be shown starting from (3.6).

Thus any run of $\mathcal{A}$ on T is also a run on $\mathfrak{M}_T^{\mathcal{S}}$. Since the acceptance criterion for a run is the same, $T \in \mathcal{L}_{time}(\mathcal{A})$ implies $\mathfrak{M}_T^{\mathcal{S}} \in \mathcal{L}_{TC}(\mathcal{A})$. □

4.3 Extending the alphabet

In this section, we provide the final pieces of our jigsaw which will help us prove our main result in the next section.

In particular, we fix a finite set Π and lift MSCs over Act to MSCs over $Act \times \Pi$. A Π-extended MSC over Act or alternately, an MSC over $Act \times \Pi$ is just an MSC $(E, \leq, \lambda)$ over Act such that $\lambda : E \to Act \times \Pi$. To be completely formal, we recall the original Definition 2.5 of an MSC over Act and modify. This is done for MSCs below, but for the remaining definitions we just describe the change required in the original definition rather than redefine completely.

Definition 4.11. *An* Π-extended MSC *over* Act *or an* MSC over $(Act \times \Pi)$ *is a finite* $(Act \times \Pi)$*-labeled poset* $M = (E, \leq, \lambda)$ *such that Conditions (2.9–2.12) are satisfied.*

We note that the definition of boundedness can be immediately adapted to this setting by ignoring the extra labeling.

Equivalences over the extended alphabet

We lift the MPA and MSO definitions to include the additional alphabet. Since, such a lift is purely syntactical, we preserve the validity of the theorems mentioned in Section 2.2.5.

We define an MPA $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, \Delta, F)$ over $Act \times \Pi$ as in Definition 2.10 with the only change being that, the transition relation on component $\mathcal{A}_p$ becomes $\to_p \subseteq (S_p \times Act_p \times \Pi \times \Delta \times S_p)$. Then observe that we have the usual semantics (from Section 2.2.3) for a run of this automaton on a Π-extended MSC over Act. $\mathcal{L}_{MSC}(\mathcal{A})$ denotes the set of all Π-extended MSCs that are accepted by $\mathcal{A}$.

Now, observe that, with the above definition, a Π-extended MSC $(E, \leq, \lambda)$ over Act is just a labeled relational structure $(E, \{\leq^E\}, \lambda)$ over $Act \times \Pi$. Thus, setting $\Sigma = Act \times \Pi$ and $\mathfrak{R} = \mathfrak{R}_{\leq}$ or $\mathfrak{R} = \mathfrak{R}_{<}$ in Definition 2.4 we obtain $\mathrm{MSO}(Act \times \Pi, \mathfrak{R}_{\leq})$ and $\mathrm{MSO}(Act \times \Pi, \mathfrak{R}_{<})$ and their semantics in terms of Π-extended MSCs over Act. The existential fragments are obvious. Also for a sentence φ from the logic, $\mathcal{L}_{MSC}(\varphi)$ denotes the set of Π-extended MSCs over Act that satisfy it.

With the above definitions, it is easy to see that we can lift the results mentioned in Section 2.2.5. We restate the theorems that are relevant to us, namely Theorem 2.15 and Theorem 2.12 as respectively,

Theorem 4.12. *Let* $\mathcal{L}$ *be a language of MSCs over* $Act \times \Pi$*. Then* $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ *for some* $\varphi \in \mathrm{EMSO}(Act \times \Pi, \mathfrak{R}_{<})$ *if and only if* $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{A})$ *for some MPA* $\mathcal{A}$ *over* $Act \times \Pi$*.*

Theorem 4.13. *Let* $B \in \mathbb{N}_{\geq 0}$*.*

1. *Let $\mathcal{L}$ be a language of existentially B-bounded MSCs over $Act \times \Pi$. Then, $\mathcal{L} = \mathcal{L}_{MSC}(\varphi)$ for some sentence $\varphi \in \mathrm{MSO}(Act \times \Pi, \mathfrak{R}_{\leq})$ if and only if $\mathcal{L} = \mathcal{L}_{MSC}(\mathcal{A})$ for some MPA $\mathcal{A}$ over $Act \times \Pi$.*

2. *There exists an MPA $\mathcal{A}$ over $Act \times \Pi$ such that $\mathcal{L}_{MSC}(\mathcal{A})$ is the set of all existentially B-bounded MSCs over $Act \times \Pi$.*

TCMSCs as extended MSCs

Let $\mathcal{S} \subseteq \mathcal{I}$ be a finite set of intervals. Then, we can consider the alphabet $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$ where $[\mathrm{TC} \dashrightarrow \mathcal{S}]$ is a finite set of partial maps from TC to $\mathcal{S}$. Recall that TC is the finite set of symbols defined in the beginning of Section 3.4.

Since $\mathcal{S}$ is finite, an ecTCMSC over $(Act, \mathcal{S})$ can be directly seen as an MSC over Γ. We make this precise by defining a function f which maps the set of ecTCMSCs over $(Act, \mathcal{S})$ to the set of MSCs over Γ:

Definition 4.14. *Let $\mathfrak{M} = (M, \tau)$ be any ecTCMSC over $(Act, \mathcal{S})$, with $M = (E, \leq, \lambda)$, $\tau : \mathrm{TC}^M \dashrightarrow \mathcal{S}$. Then $\mathrm{f}(\mathfrak{M}) = (E, \leq, \lambda')$ is an MSC over Γ with the same set of events E and partial order $\leq$. Also, $\lambda' : E \to \Gamma$ is defined for all $e \in E$, by $\lambda'(e) = (\lambda(e), g_e)$ where $g_e : \mathrm{TC} \dashrightarrow \mathcal{S}$ is such that, for all $\alpha \in \mathrm{TC}$,*

$$g_e(\alpha) = \begin{cases} \tau(e, e') & \text{if there exists } e' \in E, \text{ s.t., } (e, e') \in (\alpha^M \cap \mathrm{dom}(\tau)) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Again, we recall that given $e \in E$ and $\alpha \in \mathrm{TC}$, there can exist at most one event $e' \in E$ such that $(e, e') \in \alpha^M$.

Proposition 4.15. *The* f *defined above is an injective map, i.e, if $\mathrm{f}(\mathfrak{M}_1) = \mathrm{f}(\mathfrak{M}_2)$ then $\mathfrak{M}_1 = \mathfrak{M}_2$ for ecTCMSCs $\mathfrak{M}_1, \mathfrak{M}_2$ over $(Act, \mathcal{S})$.*

Proof. Let $\mathfrak{M}_1 = (M_1, \tau_1)$ and $\mathfrak{M}_2 = (M_2, \tau_2)$ with $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$. Then $\mathrm{f}(\mathfrak{M}_1) = (E^1, \leq^1, \lambda_1')$ and $\mathrm{f}(\mathfrak{M}_2) = (E^2, \leq^2, \lambda_2')$ are such that for all $e^1 \in E^1$, $\lambda_1'(e^1) = (\lambda_1(e^1), g^1(e^1))$ and for all $e^2 \in E^2$, $\lambda_2'(e^2) = (\lambda_2(e^2), g^2(e^2))$ where g^1, g^2 are as defined above.

First, $\mathrm{f}(\mathfrak{M}_1) = \mathrm{f}(\mathfrak{M}_2)$ implies that $E^1 = E^2$, $\leq^1 = \leq^2$ and $\lambda_1(e) = \lambda_2(e)$ for all $e \in E^1 (= E^2)$, since $\lambda_1' = \lambda_2'$. Thus $M_1 = (E^1, \leq^1, \lambda_1) = (E^2, \leq^2, \lambda_2) = M_2$ and so we have $\alpha^{M_1} = \alpha^{M_2}$. We are left with proving that $\tau_1 = \tau_2$. Let $E = E^1 = E^2$.

Since $\lambda_1' = \lambda_2'$, for each $e \in E$, $\alpha \in \mathrm{TC}$, $g_e^1(\alpha) = g_e^2(\alpha)$. But that means that $\{e' \in E \mid (e, e') \in (\alpha^{M_1} \cap \mathrm{dom}(\tau_1))\} = \{e' \in E \mid (e, e') \in (\alpha^{M_2} \cap \mathrm{dom}(\tau_2))\}$. Thus for each $(e, e') \in \alpha^{M_1} = \alpha^{M_2}$ for some $\alpha \in \mathrm{TC}$, we have $(e, e') \in \mathrm{dom}(\tau_1)$ iff $(e, e') \in \mathrm{dom}(\tau_2)$ and then $\tau_1(e, e') = g_e^1(\alpha) = g_e^2(\alpha) = \tau_2(e, e')$. Thus, we conclude that $\mathfrak{M}_1 = \mathfrak{M}_2$ and hence f is an injective map. □

Transforming TMSO formulae

In this section we show that each TMSO formula can be rewritten as an MSO formula over the extended alphabet of guards introduced above. Let $\mathcal{S}$ be a finite set of intervals and $\Gamma = Act \times [\text{TC} \dashrightarrow \mathcal{S}]$. Then from a given TMSO formula φ, we obtain a formula $\varphi^{\mathcal{S}} \in \text{MSO}(\Gamma)$ by replacing sub-formulas of the form $P_a(x)$ by the formula:

$$\bigvee_{\{(b,g)\in\Gamma \mid b=a\}} P_{(b,g)}(x)$$

and sub-formulas of the form $\delta_\alpha(x) \in I$ by the formula:

$$\bigvee_{\{(b,g)\in\Gamma \mid g(\alpha)\subseteq I\}} P_{(b,g)}(x)$$

Note that this translation preserves the existential fragment, i.e, for instance, if $\varphi \in \text{ETMSO}(Act, \mathfrak{R}_<)$, then $\varphi^{\mathcal{S}} \in \text{EMSO}(\Gamma, \mathfrak{R}_<)$.

Now since an MSC over Γ is just a labeled relational structure over Γ, from Section 2.1.2, Definition 2.4, we obtain the semantics of a formula from this logic in terms of MSCs over Γ. Then, we can relate the ecTCMSC-language of a TMSO formula and the extended MSC language of its MSO translation as follows,

Lemma 4.16. *Let φ be a TMSO sentence and $\mathcal{S}$ be a finite set of intervals. Then for an ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$, $\mathfrak{M} \models \varphi$ if and only if $\text{f}(\mathfrak{M}) \models \varphi^{\mathcal{S}}$.*

Proof. Let $\mathfrak{M} = (M, \tau)$ be an ecTCMSC over $(Act, \mathcal{S})$ with $M = (E, \leq, \lambda)$ and $\text{f}(\mathfrak{M}) = (E, \leq, \lambda')$ as defined in the previous section. We show the lemma for any interpretation $\mathbb{I}$ by induction on structure of φ. As before, the only interesting cases are atomic and timing predicates. The others are routine deductions.

- Suppose φ is of the form $P_a(x)$ for some $a \in Act$. Then, we will show that,

  $$\mathfrak{M}, \mathbb{I} \models P_a(x) \text{ if and only if } \text{f}(\mathfrak{M}), \mathbb{I} \models \bigvee_{\{(b,g)\in\Gamma \mid b=a\}} P_{(b,g)}(x)$$

  Now, $\mathfrak{M}, \mathbb{I} \models P_a(x)$ means that $\lambda(\mathbb{I}(x)) = a$. But, $\lambda(\mathbb{I}(x)) = a$ if and only if $\lambda'(\mathbb{I}(x)) = (a, g_{\mathbb{I}(x)})$. But we have $\lambda'(\mathbb{I}(x)) = (a, g_{\mathbb{I}(x)})$ if and only if $\text{f}(\mathfrak{M}), \mathbb{I} \models P_{(a,g_{\mathbb{I}(x)})}(x)$, which holds if and only if $\text{f}(\mathfrak{M}), \mathbb{I} \models \bigvee_{\{(b,g)\in\Gamma \mid b=a\}} P_{(b,g)}(x)$. Thus, we are done.

- Suppose φ be of the form $\delta_\alpha(x) \in I$ for some $\alpha \in \text{TC}$, $I \in \mathcal{I}$. Then, we show that,

  $$\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I \text{ if and only if } \text{f}(\mathfrak{M}), \mathbb{I} \models \bigvee_{\{(b,g)\in\Gamma \mid g(\alpha)\subseteq I\}} P_{(b,g)}(x)$$

($\implies$) Let $\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I$ where $\mathbb{I}(x) = e$ for some $e \in E$. By definition, this means that there exists $e' \in E$ such that $(e, e') \in \alpha^M$ and $\tau(e, e') \subseteq I$. By the definition of $\mathrm{f}(\mathfrak{M})$, we then have $\lambda'(e) = (\lambda(e), g_e)$ with $g_e : \mathrm{TC} \dashrightarrow \mathcal{S}$ such that $g_e(\alpha) = \tau(e, e') \subseteq I$. This implies that $\mathrm{f}(\mathfrak{M}), \mathbb{I} \models P_{(\lambda(e), g_e)}(x)$ with $g_e(\alpha) \subseteq I$. Hence we have, $\mathrm{f}(\mathfrak{M}), \mathbb{I} \models \bigvee_{\{(b,g) \in \Gamma \mid g(\alpha) \subseteq I\}} P_{(b,g)}(x)$.

($\impliedby$) Conversely assume that $\mathrm{f}(\mathfrak{M}), \mathbb{I} \models P_{(b,g)}(x)$ for some $(b, g) \in \Gamma$ such that $g(\alpha) \subseteq I$ and let $\mathbb{I}(x) = e$ for some $e \in E$. Then by definition $\lambda'(e) = (b, g)$, $\lambda(e) = b$ and $g : \mathrm{TC} \dashrightarrow \mathcal{S}$ with $g(\alpha) \subseteq I$. Since $g(\alpha)$ is defined, there exists $e' \in E$ such that $(e, e') \in \mathrm{dom}(\tau) \cap \alpha^M$ and $g(\alpha) = \tau(e, e') \subseteq I$. Thus, we conclude that $\mathfrak{M}, \mathbb{I} \models \delta_\alpha(x) \in I$.

Thus, by induction on the structure of φ, we obtain that for any interpretation $\mathbb{I}$, $\mathfrak{M}, \mathbb{I} \models \varphi$ if and only if $\mathrm{f}(\mathfrak{M}), \mathbb{I} \models \varphi^{\mathcal{S}}$. Hence we conclude that for any TMSO sentence φ, we have $\mathfrak{M} \models \varphi$ if and only if $\mathrm{f}(\mathfrak{M}) \models \varphi^{\mathcal{S}}$.

□

ECMPA vs MPA over the extended alphabet

Let us fix a finite set of intervals $\mathcal{S}$ and an alphabet of guards $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$. Then, observe that any MPA $\mathcal{A}$ over Γ is itself an ECMPA which uses intervals from $\mathcal{S}$ as guards, i.e, $\mathrm{Int}(\mathcal{A}) \subseteq \mathcal{S}$.

Lemma 4.17. *Let $\mathcal{S}$ be a finite set of intervals and $\mathcal{A}$ be an MPA over $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$. Then for an ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$, $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$ implies $\mathfrak{M} \in \mathcal{L}_{TC}(\mathcal{A})$.*

Proof. The proof follows from the definitions. Let $\mathcal{A}$ be the MPA over $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$. and ecTCMSC $\mathfrak{M} = (M, \tau)$ over $(Act, \mathcal{S})$ with $M = (E, \leq, \lambda)$ and $\mathrm{f}(\mathfrak{M}) = (E, \leq, \lambda')$, such that, $\lambda'(e) = (\lambda(e), g_e)$, for $g_e : \mathrm{TC} \dashrightarrow \mathcal{S}$ where $g_e(\alpha) = \tau(e, \widetilde{e})$ if there exists $\widetilde{e} \in E$ such that $(e, \widetilde{e}) \in \alpha^M \cap \mathrm{dom}(\tau)$.

Let $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$. That is, there exists a run r of the (extended) MPA $\mathcal{A}$ on the MSC $\mathrm{f}(\mathfrak{M})$ which is accepting, i.e, reaches a final state. As defined in Section 2.2.3, $r : E \to \bigcup_{p \in Proc} S_p$ is a map such that, for all $e, e' \in E$, $e <_{pq} e'$, $(p, q) \in Ch$, there exists $d \in \Delta$ such that,

$$(r^-(e), \lambda'(e), d, r(e)) \in \to_p \text{ and } (r^-(e'), \lambda'(e'), d, r(e')) \in \to_q$$

where, we recall that $r^- : E \to \bigcup_{p \in Proc} S_p$ is the map: For event $e \in E_p$, if $\exists e' \in E_p$ such that $e' <_{pp} e$, then $r^-(e) = r(e')$ and $r^-(e) = \iota_p$ otherwise.

We claim that r is a run of ECMPA $\mathcal{A}$ on the ecTCMSC $\mathfrak{M}$. That is, for all $e, e' \in E$, $e <_{pq} e'$ for some $(p, q) \in Ch$, there are $g, g' \in [\mathrm{TC} \dashrightarrow \mathcal{I}]$ and a control message $d \in \Delta$ such that Conditions (3.2–3.4) of Section 3.4 hold.

Since $\lambda'(e) = (\lambda(e), g_e)$, Condition (3.2) follows immediately. Again, by the definition of λ', $g_e \in [\mathrm{TC} \dashrightarrow \mathcal{S}]$ has the property that for all $\alpha \in \mathrm{dom}(g_e)$, there exists $\widetilde{e} \in E$ such that $(e, \widetilde{e}) \in \alpha^M \cap \mathrm{dom}(\tau)$ and $g_e(\alpha) = \tau(e, \widetilde{e'})$. Thus Condition (3.3) is satisfied. Similarly, $g_{e'}$ satisfies Condition (3.4). Thus, r is a run of $\mathcal{A}$ on $\mathfrak{M}$. It is accepting since the acceptance condition is the same and depends on reaching the final state. Thus we can conclude that $\mathfrak{M} \in \mathcal{L}_{TC}(\mathcal{A})$. □

Lemma 4.18. *Let $\mathcal{S}$ be a proper finite set of intervals and $\mathcal{A}$ be an MPA over $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$. Then, for any TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$, there exists an ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$ such that $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$ and T realizes $\mathfrak{M}$.*

Proof. We begin with an accepting run r of $\mathcal{A}$ on TMSC $T = (M, t)$. Thus, for all $e, e' \in E$, $e <_{pq} e'$ for some $(p, q) \in Ch$, there are $g_e, g_{e'} \in [\mathrm{TC} \dashrightarrow \mathcal{I}]$ and $d \in \Delta$ such that Conditions (3.2,3.5–3.6) hold. But since $\mathrm{Int}(\mathcal{A}) \subseteq \mathcal{S}$, we obtain $g_e, g_{e'} \in [\mathrm{TC} \dashrightarrow \mathcal{S}]$.

We recall that by Condition (3.5), for all $\alpha \in \mathrm{dom}(g_e)$ there exists $\widetilde{e} \in E$ such that $(e, \widetilde{e}) \in \alpha^M$ and $|t(e) - t(\widetilde{e})| \in g_e(\alpha)$. Similarly from Condition (3.6), for all $\alpha \in \mathrm{dom}(g_{e'})$ there exists $\widetilde{e'} \in E$ such that $(e', \widetilde{e'}) \in \alpha^M$ and $|t(e') - t(\widetilde{e'})| \in g_{e'}(\alpha)$. Using these partial maps above, we define another partial function $\tau \in [\mathrm{TC}^M \dashrightarrow \mathcal{S}]$ as

$$\tau(e_1, e_2) = \begin{cases} g_{e_1}(\alpha) & \text{if } \exists\ \alpha \in \mathrm{TC} \text{ s.t } (e_1, e_2) \in \alpha^M \text{ and } \alpha \in \mathrm{dom}(g_{e_1}) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Observe that τ is well-defined. If $\tau(e_1, e_2) = g_{e_1}(\alpha)$ and $\tau(e_1, e_2) = g_{e_1}(\alpha')$ for some $e_1, e_2 \in E, g_{e_1} \in [\mathrm{TC} \dashrightarrow \mathcal{S}], \alpha, \alpha' \in \mathrm{TC}$, then $|t(e_1) - t(e_2)| \in g_{e_1}(\alpha') \cap g_{e_1}(\alpha)$. Now $\mathcal{S}$ is a proper interval set implies that $g_{e_1}(\alpha) = g_{e_1}(\alpha')$.

Now using the above map we define an ecTCMSC over $(Act, \mathcal{S})$ as $\mathfrak{M} = (M, \tau)$. T realizes $\mathfrak{M}$ by definition, since for all $(e_1, e_2) \in \mathrm{dom}(\tau)$, $|t(e_1) - t(e_2)| \in g_{e_1} = \tau(e_1, e_2)$. We are done if we show that $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$. But this follows since r is itself a run of $\mathcal{A}$ over $\mathrm{f}(\mathfrak{M})$. It is enough to observe that $\lambda'(e) = (\lambda(e), g_e)$ where g_e is the partial map given above from the run on the TMSC. Then, for all $e, e' \in E$ such that $e <_{pq} e'$ for some $(p, q) \in Ch$, there exists $d \in \Delta$ such that $(r^-(e), \lambda'(e), d, r(e)) \in \rightarrow_p$, $(r^-(e'), \lambda'(e'), d, r(e')) \in \rightarrow_q$. Thus r is an accepting run of $\mathcal{A}$ on $\mathrm{f}(\mathfrak{M})$ (since acceptance depends only on reaching a final state).

□

4.4 Equivalence between ECMPA and ETMSO over TMSCs

We are now in a position to be able to state and prove the first main theorem of this chapter. We show that there is a effective equivalence between ECMPAs and TMSOs over TMSCs.

Theorem 4.19. *Let $\mathcal{L}$ be a set of TMSCs over Act. Then, the following are equivalent:*

1. *There is an ECMPA $\mathcal{A}$ such that $\mathcal{L}_{time}(\mathcal{A}) = \mathcal{L}$.*
2. *There is $\varphi \in \mathrm{ETMSO}(Act, \mathfrak{R}_{\leqslant})$ such that $\mathcal{L}_{time}(\varphi) = \mathcal{L}$.*

This equivalence is constructive in the sense that we can explicitly construct the ETMSO from the ECMPA and vice versa.

The construction of an ETMSO formula from an ECMPA follows the similar constructions applied, for example, to finite and asynchronous automata. In addition, we have to cope with the timing predicate. Assume that $g : \mathrm{TC} \dashrightarrow \mathcal{I}$ is such a guard occurring on a local transition of the given ECMPA. To ensure that the timing constraints that come along with g are satisfied we use the formula $\bigwedge_{\alpha \in \mathrm{dom}(g)} \delta_\alpha(x) \in g(\alpha)$.

The difficult part is the construction of an ECMPA from an ETMSO formula. The basic idea is to reduce this to an analogous untimed case, which has also been applied in the settings of words and traces [34,35]. For this, we use the connection we have established in the previous section between TMSO and ordinary MSO logic without timing predicate.

Usually, the untimed formalisms need to be parameterized by a finite alphabet, so that we can speak of structures whose labelings are extended with this alphabet. Hence, in our framework, we need to find a finite abstraction of the infinite set of possible time stamps. For this, we move from T-MSCs to TC-MSCs over a finite alphabet, using the converse of Lemmas 4.2 and 3.14 in terms of Lemmas 4.9 and 4.10, respectively. The latter two lemmas finally allow us to establish a translation of ETMSO formulas into ECMPAs.

Proof. (1 $\Longrightarrow$ 2) Given an ECMPA $\mathcal{A}$, we now construct $\varphi \in \mathrm{ETMSO}(Act, \mathfrak{R}_{\leqslant})$ such that $\mathcal{L}_{time}(\mathcal{A}) = \mathcal{L}_{time}(\varphi)$.

Let $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, Act, \Delta, F)$ be the ECMPA at hand with $\mathcal{A}_p = (S_p, \iota_p, \rightarrow_p)$. For any local state $s \in S = \bigcup_{p \in Proc} S_p$, we introduce a second order variable X_s. The formula we are targeting guesses a run of $\mathcal{A}$ in terms of an assignment of events to the variables $(X_s)_{s \in S}$, i.e., an interpretation that assumes an event e to be contained in X_s, stands for a run assigning e to s. Accordingly, $(X_s)_{s \in S}$ needs to be a partition of all the events of an MSC. This can easily be required by a first order formula $\mathit{Partition}((X_s)_{s \in S})$ (with free variables $(X_s)_{s \in S}$):

$$\mathit{Partition}((X_s)_{s \in S}) = \Big(\forall x \bigvee_{s \in S} (x \in X_s)\Big) \wedge \Big(\forall x \bigwedge_{s,s' \in S, s \neq s'} \neg(x \in X_s \wedge x \in X_{s'})\Big).$$

We now define some further macros. For a synchronization message $d \in \Delta$,

$$\mathit{Trans}_d(x,(X_s)_{s\in S}) =$$

$$\bigvee_{\substack{p\in \mathit{Proc}\\(s,g,a,d',s')\in\to_p\\ d'=d}} \Big(P_a(x)\wedge x\in X_{s'}\wedge \big[\exists y\ (y\lessdot_{pp} x\wedge y\in X_s)\big]\wedge \bigwedge_{\alpha\in\mathrm{dom}(g)}(\delta_\alpha(x)\in g(\alpha))\Big)$$

$$\vee \bigvee_{\substack{p\in \mathit{Proc}\\(s,g,a,d',s')\in\to_p\\ d'=d\\ s=\iota_p}} \Big(P_a(x)\wedge x\in X_{s'}\wedge \big[\neg\exists y,\ y\lessdot_{pp} x\big]\wedge \bigwedge_{\alpha\in\mathrm{dom}(g)}(\delta_\alpha(x)\in g(\alpha))\Big)$$

This formula describes that, under the assignment $(X_s)_{s\in S}$, the execution of x actually corresponds to a local transition of $\mathcal{A}$ that communicates $d\in\Delta$.

Moreover, for a global state $\overline{s}=(s_p)_{p\in \mathit{Proc}}\in\prod_{s\in \mathit{Proc}} S_p$, we set,

$$\mathit{Final}_{\overline{s}}((X_s)_{s\in S}) =$$

$$\bigvee_{\substack{\mathit{Proc}'\subseteq \mathit{Proc}\\ \forall p\in \mathit{Proc}':\ s_p=\iota_p}} \left(\bigwedge_{p\in \mathit{Proc}\setminus \mathit{Proc}'} \exists x\big(\mathit{max}_p(x)\wedge x\in X_{s_p}\big)\wedge \bigwedge_{p\in \mathit{Proc}'}\neg\exists x\Big(\bigvee_{a\in \mathit{Act}_p} P_a(x)\Big)\right)$$

Hereby, given a process $p\in \mathit{Proc}$, $\mathit{max}_p(x)=\bigvee_{a\in \mathit{Act}_p} P_a(x)\wedge\neg\exists y,\ x\lessdot_{pp} y$. Moreover, observe that Proc' comprises those processes that are assumed not to move. Hence, $\mathit{Final}_{\overline{s}}((X_s)_{s\in S})$ formulates that the run described by $(X_s)_{s\in S}$ ends up in the global state $\overline{s}$.

We are now prepared to give the formula φ with $\mathcal{L}_{\mathit{time}}(\varphi)=\mathcal{L}_{\mathit{time}}(\mathcal{A})$. Namely,

$$\begin{aligned}\varphi = \exists (X_s)_{s\in S}\ & \mathit{Partition}((X_s)_{s\in S})\\ \wedge\ & \forall x\forall y\Big(\bigvee_{(p,q)\in \mathit{Ch}} x<_{pq} y\to \bigvee_{d\in\Delta}\big(\mathit{Trans}_d(x,(X_s)_{s\in S})\wedge \mathit{Trans}_d(y,(X_s)_{s\in S})\big)\Big)\\ \wedge\ & \bigvee_{\overline{s}\in F}\mathit{Final}_{\overline{s}}((X_s)_{s\in S})\end{aligned}$$

This concludes the proof in one direction.

(2 $\Longrightarrow$ 1) For the other direction, we now show that, given a sentence $\varphi\in \mathrm{ETMSO}(\mathit{Act},\mathfrak{R}_<)$, we can construct an ECMPA $\mathcal{A}$ such that $\mathcal{L}_{\mathit{time}}(\mathcal{A})=\mathcal{L}_{\mathit{time}}(\varphi)$.

Let φ be the given $\mathrm{ETMSO}(\mathit{Act},\mathfrak{R}_<)$ formula and let $\mathcal{S}$ be a proper interval set which refines $\mathrm{Int}(\varphi)$. Fix an alphabet $\Gamma=\mathit{Act}\times[\mathrm{TC}\dashrightarrow\mathcal{S}]$. Thus, we have $\varphi^{\mathcal{S}}\in\mathrm{EMSO}(\Gamma,\mathfrak{R}_<)$ using the translation from Section 4.3.

Then, by Theorem 4.12 we obtain an MPA $\mathcal{A}$ over $\mathit{Act}\times[\mathrm{TC}\dashrightarrow\mathcal{S}]$ such that $\mathcal{L}_{\mathit{MSC}}(\mathcal{A})=\mathcal{L}_{\mathit{MSC}}(\varphi^{\mathcal{S}})$. But then, from Definition 3.13, we can infer that $\mathcal{A}$ is an

ECMPA with $\mathrm{Int}(\mathcal{A}) \subseteq \mathcal{S}$. Now we claim that this is, in fact, the ECMPA that we require,

Claim 4.20. *We will show that* $\mathcal{L}_{time}(\varphi) = \mathcal{L}_{time}(\mathcal{A})$.

Proof. ($\subseteq$) Let $T \models \varphi$. Since $\mathcal{S}$ is a proper interval set that refines $\mathrm{Int}(\varphi)$, we can apply Lemma 4.9 to get $\mathfrak{M}_T^{\mathcal{S}} \models \varphi$. Since, $\mathcal{S}$ is proper, it is finite and hence we use Lemma 4.16 to obtain $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \models \varphi^{\mathcal{S}}$. But $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}}) = \mathcal{L}_{MSC}(\mathcal{A})$ and so we have $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \in \mathcal{L}_{MSC}(\mathcal{A})$. As $\mathcal{S}$ is finite and $\mathfrak{M}_T^{\mathcal{S}}$ is a ecTCMSC over $(Act, \mathcal{S})$, we can then use Lemma 4.17 to conclude that $\mathfrak{M}_T^{\mathcal{S}} \in \mathcal{L}_{TC}(\mathcal{A})$. Then, since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, we can use Lemma 3.14 to obtain $T \in \mathcal{L}_{time}(\mathcal{A})$.

($\supseteq$) Let $T \in \mathcal{L}_{time}(\mathcal{A})$. Then as $\mathcal{S}$ is a proper interval set, and $\mathcal{A}$ is an MPA over $Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$, by Lemma 4.18, there exists a ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$ such that $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$ and T realizes $\mathfrak{M}$. But $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}}) = \mathcal{L}_{MSC}(\mathcal{A})$ implies $\mathrm{f}(\mathfrak{M}) \models \varphi^{\mathcal{S}}$. Now, by Lemma 4.16 we have $\mathfrak{M} \models \varphi$. Finally, since T realizes $\mathfrak{M}$, by Lemma 4.2 we conclude that $T \models \varphi$. □

Hence we have shown both directions of the result that we set out to prove in this section. □

4.5 Equivalence between ECMPA and TMSO over Bounded TMSCs

Along the lines of the proof in the previous section, we can also get a characterization of the full TMSO. However, we have to restrict to untimed-existentially-bounded TMSCs. Thus, we have,

Theorem 4.21. *Let* $B \in \mathbb{N}_{\geq 0}$ *and let* $\mathcal{L}$ *be a set of* $\exists^u$*-*B*-bounded TMSCs over* Act*. Then, the following are equivalent:*

1. *There is an ECMPA* $\mathcal{A}$ *such that* $\mathcal{L}_{time}(\mathcal{A}) = \mathcal{L}$.
2. *There is* $\varphi \in \mathrm{TMSO}(Act, \mathfrak{R}_{\leq})$ *such that* $\mathcal{L}_{time}(\varphi) = \mathcal{L}$.

Proof. (1 $\implies$ 2) Let $\mathcal{L}$ be the given set of $\exists^u$-B-bounded TMSCs over Act. Given an ECMPA $\mathcal{A}$, by Theorem 4.19, we obtain an $\mathrm{ETMSO}(Act, \mathfrak{R}_{<})$ formula φ such that $\mathcal{L}_{time}(\mathcal{A}) = \mathcal{L}_{time}(\varphi) = \mathcal{L}$. But lifting Proposition 2.14 to this setting, we can observe that for any $\mathrm{ETMSO}(Act, \mathfrak{R}_{<})$ formula, there is a $\mathrm{TMSO}(Act, \mathfrak{R}_{\leq})$ formula that accepts the same set of $\exists^u$-B-bounded TMSCs. Thus, we have the required result.

(2 $\implies$ 1) Let $B \in \mathbb{N}_{\geq 0}$ and φ be the given $\mathrm{TMSO}(Act, \mathfrak{R}_{\leq})$ formula and let $\mathcal{S}$ be a proper interval set which refines $\mathrm{Int}(\varphi)$. Fix an alphabet $\Gamma = Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$.

Thus, we have $\varphi^{\mathcal{S}} \in \mathrm{MSO}(\Gamma, \mathfrak{R}_{\leq})$ using the translation from Section 4.3. Now applying Theorem 4.13[2] to Theorem 4.13[1], we obtain a formula $\psi \in \mathrm{MSO}(\Gamma, \mathfrak{R}_{\leq})$ such that $\mathcal{L}_{MSC}(\psi)$ is the set of all existentially B-bounded MSCs over Γ.

Then, indeed $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}} \wedge \psi)$ is an existentially B-bounded set of MSCs over Γ and so, we apply Theorem 4.13[1] again to the $\mathrm{MSO}(\Gamma, \mathfrak{R}_{\leq})$ formula $\varphi^{\mathcal{S}} \wedge \psi$ to obtain an MPA $\mathcal{A}$ over Γ such that $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}} \wedge \psi) = \mathcal{L}_{MSC}(\mathcal{A})$. Now again, from Definition 3.13, $\mathcal{A}$ is an ECMPA such that $\mathcal{S}$ refines $\mathrm{Int}(\mathcal{A})$.

Claim 4.22. $\mathcal{L}_{time}(\varphi) = \mathcal{L}_{time}(\mathcal{A})$

Proof. First observe that if T is $\exists^u$-B-bounded, then for any $\mathfrak{M}$ such that T realizes $\mathfrak{M}$, $\mathrm{f}(\mathfrak{M})$ is $\exists$-B-bounded. This is true since a (untimed) linearization of a TMSC only depends on its set of events and the partial order. And it is easily seen from the definitions that the set of events and partial order of T and $\mathrm{f}(\mathfrak{M})$ are the same if T realizes $\mathfrak{M}$. Thus, they have the same set of linearizations and so T is $\exists^u$-B-bounded if and only if $\mathrm{f}(\mathfrak{M})$ is $\exists$-B-bounded.

($\Longrightarrow$) Let us start with $T \in \mathcal{L}_{time}(\varphi)$, T is $\exists^u$-B-bounded. Then, $T \models \varphi$ and by Lemma 4.9, we obtain that $\mathfrak{M}_T^{\mathcal{S}} \models \varphi$. Again since $\mathcal{S}$ is proper, it is finite and hence by Lemma 4.16 we get $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \models \varphi^{\mathcal{S}}$. Since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}})$ is $\exists$-B-bounded, and so we get $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \models \psi$. Thus we conclude that $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \in \mathcal{L}_{MSC}(\varphi^{\mathcal{S}} \wedge \psi)$. But $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}} \wedge \psi) = \mathcal{L}_{MSC}(\mathcal{A})$ implies that $\mathrm{f}(\mathfrak{M}_T^{\mathcal{S}}) \in \mathcal{L}_{MSC}(\mathcal{A})$. Then, by Lemma 4.17 we get $\mathfrak{M}_T^{\mathcal{S}} \in \mathcal{L}_{TC}(\mathcal{A})$. And finally, since T realizes $\mathfrak{M}_T^{\mathcal{S}}$, we can use Lemma 3.14 to obtain $T \in \mathcal{L}_{time}(\mathcal{A})$.

($\Longleftarrow$) Let $T \in \mathcal{L}_{time}(\mathcal{A})$. Then as $\mathcal{S}$ is a proper interval set, and $\mathcal{A}$ is an MPA over $Act \times [\mathrm{TC} \dashrightarrow \mathcal{S}]$, by Lemma 4.18, there exists an ecTCMSC $\mathfrak{M}$ over $(Act, \mathcal{S})$ such that $\mathrm{f}(\mathfrak{M}) \in \mathcal{L}_{MSC}(\mathcal{A})$ and T realizes $\mathfrak{M}$. But $\mathcal{L}_{MSC}(\varphi^{\mathcal{S}} \wedge \psi) = \mathcal{L}_{MSC}(\mathcal{A})$ implies $\mathrm{f}(\mathfrak{M}) \models \varphi^{\mathcal{S}}$ and $\mathrm{f}(\mathfrak{M})$ is $\exists^u$-B-bounded. Now, by Lemma 4.16 we have $\mathfrak{M} \models \varphi$. Finally, since T realizes $\mathfrak{M}$, by Lemma 4.2 we conclude that $T \models \varphi$. □

□

5 Checking Emptiness of ECMPAs

In this chapter, we investigate emptiness checking for ECMPAs, leading to a partial solution to the satisfiability problem for TMSO formulas, which is undecidable in its full generality. Since the MSCs are used in early protocol design, this problem is of vital interest as it allows detection of possible design failures at this stage.

Theorem 5.1. *The following problem is decidable:*

INPUT: *An ECMPA $\mathcal{A}$ and an integer $B > 0$.*

QUESTION: *Does there exist $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T has a B-bounded timed linearization?*

Then, using Theorem 4.21 we can conclude that,

Corollary 5.2. *The following problem is decidable:*

INPUT: *A TMSO formula φ and an integer $B > 0$.*

QUESTION: *Does there exist $T \in \mathcal{L}_{time}(\varphi)$ such that T has a B-bounded timed linearization?*

The rest of the chapter formulates the proof of the above theorem. Let $\mathcal{A}$ be an ECMPA and let $B > 0$. We construct a (finite) *timed automaton* $\mathcal{B}$ that accepts a timed word w over Act if and only if w is a B-bounded timed linearization of some TMSC accepted by $\mathcal{A}$. Since emptiness is decidable for finite timed automata (cf. Theorem 2.21 (1)), we are done.

The remainder of this section is dedicated to the construction of such a $\mathcal{B}$, which is done in three steps as sketched below:

- First, we address the main hurdle in simulating an ECMPA by a TA, namely, a run of a TA is totally ordered, while ECMPAs have partially ordered runs. Hence, to keep track of clock constraints used in the ECMPA, the TA needs to recover the partial order information from its runs, i.e, words. This is done using gadgets that we will define as our first step.

- Next, using these gadgets, we describe an infinite timed automaton which simulates the ECMPA.

- Though we allow infinitely many clocks and states in this intermediate construction, on any run we will see that only finitely many states and clocks are used. We show how to modify this automaton in order to get down to finitely many states and clocks thus completing our construction.

5.1 Recovering the partial order

We begin by introducing some notations that will be used to classify the set of actions. For any channel $(p, q) \in Ch$, and each $\theta \in \{!, ?\}$, $p\theta q$ denotes the set of actions $\{p\theta q(m) \in Act \mid m \in \mathcal{M}\}$. Thus, we write $a \in p!q$ (respectively, $a \in p?q$) if $a = p!q(m)$ (respectively, $p?q(m)$) for some $m \in \mathcal{M}$.

Now, recall that the partial order of an MSC can be recovered from any of its linearizations under the fifo assumption. Indeed, if $w = a_1 \dots a_n \in Act^*$ is a linearization of MSC $M = (E, \leq^M, \lambda)$ over Act, then M is isomorphic to the unique MSC $M^w = (E^w, \leq^{M^w}, \lambda^w)$ over Act, where $E^w = \{1, \dots, n\}$ (i.e., the set of positions of the word w), $\lambda^w(i) = a_i$, and $\leq^{M^w}$ is defined as the reflexive transitive closure of $\bigcup_{p,q \in Proc} <_{pq}^{M^w}$ where, for all $p \in Proc$, $<_{pp}^{M^w} = \{(i, j) \in E^w \times E^w \mid i < j$ and $\lambda^w(i), \lambda^w(j) \in Act_p\}$ and for all $(p, q) \in Ch, <_{pq}^{M^w} = \{(i, j) \in E^w \times E^w \mid \lambda^w(i) = p!q(m), \lambda^w(j) = q?p(m)$ for some $m \in \mathcal{M}$, $|\{k \leq i \mid \lambda^w(k) \in p!q\}| = \{k \leq j \mid \lambda^w(k) \in q?p\}|\}$

Therefore, we can consider the partial order relation of M to be a relation over the positions of a given linearization of M. Thus, given a linearization w of an MSC M, we can identify M with M^w and write $i \leq^M j$ for $1 \leq i, j \leq |w|$, to mean $i \leq^{M^w} j$, i.e, the isomorphic images of positions i, j are related by $\leq^M$. Similarly, we may write $i <_{pp}^M j$, $i <_{pq}^M j$, $(i, j) \in \mathrm{Prev}_a^M$ and $(i, j) \in \mathrm{Next}_a^M$ for $a \in Act$ to mean that the corresponding events are related in M, respectively by, the local-process relation $<_{pp}^M$, the message relation $<_{pq}^M$, the previous and the next occurence of a relations. In the same way, we can also write $\lambda(i)$ instead of $\lambda^w(i)$.

Note that $i \leq^M j$ implies $i \leq j$ but the converse need not be true (where $\leq$ is the usual ordering between i and j as elements of $\mathbb{N}_{>0}$). However, if $\lambda(i) = \lambda(j)$, then $i \leq j$ implies $i \leq^M j$.

Now, we go further and describe gadgets, which are deterministic finite-state automata that run on words which are linearizations of an MSC and accept if the first and last position of the word are related under the partial order. But for this, we need to restrict to B-bounded linearizations. We begin with a definition,

Definition 5.3. *Let $M = (E, \leq, \lambda)$ be an MSC over Act and $B \in \mathbb{N}_{>0}$. Then a* B-well-stamping *for M is a map $\rho : E \to \{0, \dots, B-1\}$ such that for any $e \in E$ with $\lambda(e) = p\theta q(m)$ for some $p, q, \in Proc, m \in \mathcal{M}$ and $\theta \in \{!, ?\}$, we have $\rho(e) = |{\downarrow}e \cap \bigcup_{m' \in \mathcal{M}} E_{p\theta q(m')}| \mod B$.*

Then, by the above considerations, given a linearization w of M, ρ is also a map from the positions of w to $\{0, \ldots, B-1\}$. Now, the following property is immediate from the definitions of the message relation in an MSC (Section 2.2.1) and B-boundedness (Section 2.2.2).

Proposition 5.4. *Let $w = a_1 \ldots a_n \in Act^*$ be a B-bounded linearization of an MSC M over Act and ρ be a B-well-stamping for M. Then, for $1 \leq i < j \leq n$ and $(p,q) \in Ch$, we have $i <^M_{pq} j$ if and only if the following conditions hold:*

(a) $a_i \in p!q$, $a_j \in q?p$,

(b) $\rho(i) = \rho(j)$

(c) for all $k, i \leq k < j$, $a_k \in q?p$ implies $\rho(k) \neq \rho(i)$.

Proof. First, for any $i \in \{1, \ldots, n\}$, if $a_i = p\theta q(m)$ for some $\theta \in \{!, ?\}, m \in \mathcal{M}$, we fix the notation $\Downarrow i = \big(\downarrow i \cap \bigcup_{m' \in \mathcal{M}} E_{p\theta q(m')}\big) = \{i' \leq i \mid \lambda(i') \in p\theta q\}$. Now, the proof follows once we recall the relevant definitions in this setting.

(1) For $1 \leq i < j \leq n$, $i <^M_{pq} j$ iff $\lambda(i) = p!q(m)$, $\lambda(j) = q?p(m)$ for some $m \in \mathcal{M}$ and $|\Downarrow i| = |\Downarrow j|$.

(2) For all $1 \leq i \leq n$, $\rho(i) = |\Downarrow i| \mod B$.

(3) w is B-bounded iff for all $1 \leq \ell \leq n, (p,q) \in Ch$, $|\{i' \mid i' \leq \ell, \lambda(i') \in p!q\}| - |\{j' \mid j' \leq \ell, \lambda(j') \in q?p\}| \leq B$.

Now, we prove the proposition. First, let $i <^M_{pq} j$. Then, by (1), (a) holds and $|\Downarrow i| = |\Downarrow j| = \alpha \cdot B + b$. Then, $b = \rho(i) = \rho(j)$ and so (b) holds. Now, suppose (c) did not hold. Then there exists k, $i \leq k < j$ such that $a_k \in q?p$ and $\rho(k) = \rho(i) = \rho(j)$. Then, by (2), $|\Downarrow k| = \alpha' \cdot B + b$. But then $\Downarrow k \subseteq \Downarrow j$ and $j \in \Downarrow j$, $j \notin \Downarrow k$ implies that $|\Downarrow j| > |\Downarrow k| \implies \alpha \cdot B + b > \alpha' \cdot B + b \implies (\alpha - \alpha') \cdot B > 0$. Thus $|\Downarrow j| - |\Downarrow k| \geq B$ and so $|\Downarrow i| - |\Downarrow k| \geq B$. But now, $i < k$ and $\lambda(k) \in q?p$ implies that $|\Downarrow k| \geq |\{i' \leq i \mid \lambda(i') \in q?p\}| + 1$ and so we can conclude that $|\Downarrow i| - |\{i' \leq i \mid \lambda(i') \in q?p\}| > B$. This means $|\{i' \leq i \mid \lambda(i') \in p!q\}| - |\{i' \leq i \mid \lambda(i') \in q?p\}| > B$ and so by (3) (in particular, by letting $\ell = i$) it contradicts the B-boundedness of w. Thus (c) also holds.

Conversely, let conditions (a), (b) and (c) be true. Then, (a) and (b) together imply that $|\Downarrow i| \mod B = |\Downarrow j| \mod B$. That is $|\Downarrow i| = \alpha \cdot B + b$ and $|\Downarrow j| = \alpha' \cdot B + b$ for some $\alpha, \alpha' \in \mathbb{N}$. We now show that $\alpha = \alpha'$. First, observe that between i and j, there can be at most $B-1$ actions labelled $q?p(m')$ for some $m' \in \mathcal{M}$. Otherwise, we can find an event k, $i < k < j$ such that $a_k \in q?p$ and $|\Downarrow k| = (\alpha' - 1)B + b$. And for this k, $\rho(k) = \rho(j)$ which contradicts condition (c). Also, we have $i < j$ and $a_j \in q?p$. Thus, $(\alpha' - 1) \cdot B + b < |\{j' \leq i \mid a_i \in q?p\}| < \alpha' \cdot B + b$.

Now, if $\alpha > \alpha'$, then $|\{i' \leq i \mid \lambda(i') \in p!q\}| - |\{j' \leq i, \lambda(j') \in q?p\}| > \alpha \cdot B - \alpha' \cdot B > B$ which contradicts B-boundedness of w by (3). Else if $\alpha' > \alpha$, then $|\{j' \leq i, \lambda(j') \in q?p\}| - |\{i' \leq i \mid \lambda(i') \in p!q\}| > (\alpha' - 1 - \alpha) \cdot B \geq 0$ which is a contradiction since in a linearization, at no point can we have that the number of receives exceeds the number of sends. Thus we conclude that $\alpha = \alpha'$.

Now since number of $p!q$ events below i is equal to the number of $q?p$ events below j, we conclude by fifo property that j is the matching receive for i, i.e, $a_i = p!q(m)$ and $a_j = q?p(m)$ for some $m \in \mathcal{M}$. Thus, by (1), $i <^M_{pq} j$.

□

5.1.1 Constructing the gadgets

The first gadget we build is a deterministic finite-state automaton $\mathcal{C}^{\leq}$ over $Act \times \{0, \ldots, B-1\}$, that detects if the first and the last position of a word, provided it corresponds to some factor of a B-bounded MSC linearization, are related with respect to the associated partial ordering. In what follows, we let *Send* denote the set of symbols $\{p!q \mid (p,q) \in Ch\}$.

We define $\mathcal{C}^{\leq} = (Q^{\leq}, \delta^{\leq}, s_0^{\leq}, F^{\leq})$ where,

- A state of $Q^{\leq}$ is a triple of the form (P, O, f) where $f \in \{0,1\}$, $P \in 2^{Proc}$ and $O : Send \dashrightarrow \{0, \ldots, B-1\}$ such that if $p!q \in \mathrm{dom}(O)$ for some $(p,q) \in Ch$, then $p \in P$.
- The set of transitions is $\delta^{\leq} \subseteq Q^{\leq} \times Act \times \{0, \ldots, B-1\} \times Q^{\leq}$.
- The initial state is $s_0^{\leq} = (\emptyset, \emptyset, 0) \in Q^{\leq}$.
- The set of final states $F^{\leq} = \{(P, O, f) \in Q^{\leq} \mid f = 1\}$.

The triple (P, O, f) represents the information we need to recover the partial order. The set P contains the processes in the future partial view, O indicates the stamp/numbering of each "open send", and f will be 1 if and only if the first and the last position are in fact related as desired. Note that, letting $n = |Proc|$, we have

$$|Q^{\leq}| = 2 \cdot \sum_{k=0}^{n} \left(\binom{n}{k} \cdot (B+1)^{k(n-1)} \right) = B^{\mathcal{O}(n^2)} \qquad (\text{for } B \geq 2). \tag{5.1}$$

Now, we define the transition relation as $((P, O, f), (a, \beta), (P', O', f')) \in \delta^{\leq}$ if

the following hold:

$$P' = \begin{cases} P \cup \{p\} & \text{if } a \in Act_p \text{ and either } (P = \emptyset) \text{ or } (\exists q \in P \text{ such that,} \\ & a \in p?q \text{ and } O(q!p) = \beta) \\ P & \text{otherwise} \end{cases}$$

$$O' = \begin{cases} O[p!q \mapsto \beta] & \text{if } a \in p!q \notin \text{dom}(O) \text{ for some } (p,q) \in Ch \text{ and} \\ & \text{either } p \in P \text{ or } P = \emptyset \\ O & \text{otherwise} \end{cases}$$

$$f' = \begin{cases} 1 & \text{if } a \in Act_p \text{ for } p \in P' \\ 0 & \text{otherwise} \end{cases}$$

Of course, a run of $\mathcal{C}^{\leq}$ on $(a_1, \beta_1) \dots (a_n, \beta_n) \in (Act \times \{0, \dots, B-1\})^*$ is a sequence $s_0^{\leq} \xrightarrow{(a_1,\beta_1)} \dots \xrightarrow{(a_n,\beta_n)} s_n^{\leq}$, where $s_i^{\leq} \in Q^{\leq}$ for each $i \in \{0, \dots, n\}$ and $(s_i^{\leq}, a_{i+1}, \beta_{i+1}, s_{i+1}^{\leq}) \in \delta^{\leq}$ for all $i \in \{0, \dots, n-1\}$ and is accepting if $s_n^{\leq} \in F^{\leq}$. Then, $\mathcal{L}(\mathcal{C}^{\leq})$ denotes the set of words over $Act \times \{0, \dots, B-1\}$ having accepting runs. Note that with this definition, $\mathcal{C}^{\leq}$ has a run (which may be accepting or non-accepting) on every word of $(Act \times \{0, \dots, B-1\})^*$. For linearizations, $\mathcal{C}^{\leq}$ has the desired property:

Lemma 5.5. *Let $w = a_1 \dots a_n \in Act^*$ be a B-bounded linearization of an MSC M over Act and ρ be a B-well-stamping for M. Then, for $1 \leq i \leq j \leq n$, $(a_i, \rho(i)) \dots (a_j, \rho(j)) \in \mathcal{L}(\mathcal{C}^{\leq})$ if and only if $i \leq^M j$.*

Proof. Let us fix $1 \leq i \leq n$ and let $a_i \in Act_p$ for some $p \in Proc$. For each j such that $i \leq j \leq n$, we denote $w_{ij}^{\rho} = (a_i, \rho(i)) \dots (a_j, \rho(j))$. Then for each j, $i \leq j \leq n$, there is a run r_{ij} of $\mathcal{C}^{\leq}$ on w_{ij}^{ρ}, namely, $s_0^{\leq} \xrightarrow{(a_i,\rho(i))} (P_i, O_i, f_i) \dots \xrightarrow{(a_j,\rho(j))} (P_j, O_j, f_j)$ where $s_0^{\leq} = (\emptyset, \emptyset, 0)$ and for each $k \in \{i, \dots, j\}$, $(P_k, O_k, f_k) \in Q^{\leq}$ is given from the definition of $\mathcal{C}^{\leq}$. Indeed each r_{ij} is a prefix of $r_{ij'}$ for $j \leq j' \leq n$. For simplicity of notation, in what follows, we let $P_{i-1} = \emptyset, O_{i-1} = \emptyset, f_{i-1} = 0$, i.e, $s_0^{\leq} = (P_{i-1}, O_{i-1}, f_{i-1})$.

Now by definition of $\mathcal{C}^{\leq}$, $P_{i-1} = \emptyset, a_i \in Act_p$ implies that $P_i = \{p\}$. Once a process gets into P it is never removed, and so, $p \in P_j$ for all $i \leq j \leq n$. Also note that for each j, $i \leq j \leq n$, r_{ij} is an accepting run if and only if $f_j = 1$ i.e, if and only if $a_j \in Act_q$ for some $q \in Proc$ and $q \in P_j$.

We now distinguish two cases. First, suppose $a_j \in Act_p$, then we already have $p \in P_j$ and so r_{ij} is accepting. Also in this case, a_i, a_j belong to the same process p and so $i \leq_{pp}^M j$ which implies $i \leq^M j$. Thus,

$$\forall j \in \{i, \dots, n\}, a_j \in Act_p \implies (r_{ij} \text{ is accepting}) \wedge (i \leq^M j). \tag{5.2}$$

On the other hand, for all $j \in \{i+1, \ldots, n\}$ such that $a_j \in Act_q$ for some $q \neq p$, we will show that r_{ij} is an accepting run if and only if $i \leq^M j$ which completes the proof of the lemma. We proceed by induction on $j - i$.

The base case is when $j - i = 1$, i.e $w^\rho_{ij} = (a_i, \rho(i))(a_j, \rho(j))$. Then, $P_{j-1} = P_i = \{p\}$. Now, since $a_j \in Act_q$, r_{ij} is accepting if and only if $q \in P_j$. By definition of $\mathcal{C}^\leq$, this happens if and only if $a_j \in q?p$ and $O_{j-1}(p!q) = \rho(j)$. But since $j - 1 = i$ we have $p!q \in \mathrm{dom}(O_i)$ and along with $O_{i-1} = \emptyset$, this implies that $a_i \in p!q$ and $O_i(p!q) = \rho(i)$. Thus we can conclude that $q \in P_j$ if and only if $a_i \in p!q, a_j \in q?p$ and $\rho(i) = \rho(j)$ which by Proposition 5.4 happens if and only if $i <^M_{pq} j$ (by noting that there is no event between i and j). But again $i + 1 = j$ and i, j are not on the same process, thus, $i <^M_{pq} j$ if and only if $i \leq^M j$, which completes the proof of the base case.

Now, suppose $j - i > 1$. Then, r_{ij} is accepting implies $q \in P_j$. But since $q \notin P_i = \{p\}$, there exists j' for $i < j' \leq j$ such that $q \notin P_{j'-1}, q \in P_{j'}$. By the definition of $\mathcal{C}^\leq$, since $P_{j'-1} \neq \emptyset$, there is $p' \in Proc$ such that $a_{j'} \in q?p'$ and $O_{j'-1}(p'!q) = \rho(j')$. But since $O_{i-1} = \emptyset$, there is k, $i \leq k < j'$ such that $p'!q \in \mathrm{dom}(O_k)$ and $p'!q \notin \mathrm{dom}(O_{k-1})$. Then at k, $a_k \in p'!q$ and $O_k(p'!q) = \rho(k)$ and either $p' \in P_{k-1}$ or $k = i$ and $P_{k-1} = \emptyset$. In both cases, $p' \in P_k$ since $a_k \in Act_{p'}$. Thus, in fact $f_k = 1$ and so r_{ik} is an accepting run of $\mathcal{C}^\leq$ on w^ρ_{ik}.

Now, if $p' = p$, i.e, $a_k \in Act_p$, by (5.2), we conclude that $i \leq^M k$. Otherwise, $a_k \in Act_{p'}$, $p' \neq p$ and since, $k < j' \leq j$, we can apply the induction hypothesis to conclude that $i \leq^M k$. Now, the values in O, once defined, are never modified and so $O_k(p'!q) = O_{j'-1}(p'!q)$ which implies $\rho(k) = \rho(j')$. Also for any j'', $k \leq j'' < j'$, if $a_{j''} \in q?p'$ and $\rho(j'') = \rho(j')$, then at j'', we have $O_{j''}(p'!q) = \rho(k) = \rho(j'')$ and so $q \in P_{j''}$. This is a contradiction since we assumed that $q \in P_{j'-1}$. Thus, by Proposition 5.4, $k <^M_{p'q} j'$. Also $a_{j'}, a_j \in Act_q$ implies that $j' \leq^M_{qq} j$. Thus we have $i \leq^M k <^M_{p'q} j' \leq^M_{qq} j$ and so by definition of $\leq^M$, we conclude that $i \leq^M j$.

Conversely, let $i \leq^M j$, be such that i is a p-event and j is a q-event for $p \neq q$. Then let j' be the earliest event on process q which is related to i. This implies that $a_{j'}$ is a receive action from some process, say $p' \in Proc$. In other words, there exists k, j, such that $i \leq k < j' \leq j$ and $i \leq^M k <^M_{p'q} j' \leq^M_{qq} j$ and for all j'' with $j'' <^M_{qq} j'$, $i \not\leq^M j''$, where $a_k = p'!q(m)$ and $a_{j'} = q?p'(m)$ for some $p' \in Proc$, $m \in \mathcal{M}$.

Then, if $p' = p$, then by (5.2), and otherwise by induction hypothesis, r_{ik} is an accepting run of $\mathcal{C}^\leq$ on w_{ik}. Thus $f_k = 1$ and so $p' \in P_k$. But since, $a_k \in p'!q$, we find that either $P_k = P_{k-1}$ or $k = i$. Thus either $p' \in P_{k-1}$ or $P_{k-1} = \emptyset$. Also, $p'!q \notin \mathrm{dom}(O_{k-1})$, for otherwise, we can find k', $i \leq k' < k$ with $a_{k'} = p'!q$, whose corresponding receive contradicts the minimality of j' under fifo condition.

So by definition of O, we have $O_k(p'!q) = \rho(k)$ and by Proposition 5.4, $\rho(k) = \rho(j')$. Thus, by definition of $\mathcal{C}^\leq$, we can conclude that $P_{j'} = P_{j'-1} \cup \{q\}$. Thus, $q \in P_{j'} \subseteq P_j$ and along with $a_j \in Act_q$, we get that $f_j = 1$ and so r_{ij} is an accepting

run.

□

An ECMPA uses Prev_a and Next_a in the guards, which along a run constrain the previous and next occurence of an action a respectively. Hence, we need to recover not only the general partial order relation but also these previous and next occurence relations from the linearization. For this, we build two more gadgets using the gadget described above.

We begin by defining a deterministic finite-state automaton $\mathcal{C}^\triangleleft = (Q^\triangleleft, \delta^\triangleleft, s_0^\triangleleft, F^\triangleleft)$, which we refer to as the $\mathcal{C}^\triangleleft$ *gadget* and use to recover the previous occurence relation. Its state space is given by $Q^\triangleleft = Q^\leq \times Q^\leq \times Act$. Thus, $|Q^\triangleleft| = |Q^\leq|^2 |Proc|(|Proc|-1)|\mathcal{M}| = B^{\mathcal{O}(|Proc|^2)}$ by Equation 5.1 (we always consider the number of message contents $|\mathcal{M}|$ to be a constant). Now, the idea is that the first component, mimicking the automaton $\mathcal{C}^\leq$, is started when reading the first occurence of an action a, which is henceforth stored in the third component. The second component is run if and when a is executed for the second time. Any further event that is related in the partial order to the first occurence of a but not to the second occurence matches the Prev_a relation so that $F^\triangleleft = F^\leq \times (Q^\leq \setminus F^\leq) \times Act$. We let $s_0^\triangleleft = (s_0^\leq, s_0^\leq, b)$ for some arbitrary action $b \in Act$. Finally, for any $(s_1, s_2, a) \in Q^\triangleleft$, $b \in Act$, and $n \in \{0, \dots, B-1\}$, we set $\delta^\triangleleft((s_1, s_2, a), (b, n)) =$

$$\begin{cases} (\delta^\leq(s_1, (b, n)), s_2, b) & \text{if } s_1 = s_0^\leq \\ (\delta^\leq(s_1, (b, n)), s_2, a) & \text{if } s_1 \neq s_0^\leq,\ s_2 = s_0^\leq \text{ and } b \neq a \\ (\delta^\leq(s_1, (b, n)), \delta^\leq(s_2, (b, n)), a) & \text{otherwise.} \end{cases}$$

Similarly, the $\mathcal{C}^\triangleright$ *gadget* is defined as $\mathcal{C}^\triangleright = (Q^\triangleright, \delta^\triangleright, s_0^\triangleright, F^\triangleright)$ with $Q^\triangleright = Q^\leq \times 2^{Act} \times \{0, 1\}$. Again by Equation 5.1, $|Q^\triangleright| \leq 2|Q^\leq|\ 2^{|Proc|^2|\mathcal{M}|} = B^{\mathcal{O}(|Proc|^2)}$. The idea here is that the second component of a state keeps track of the actions we have seen so far in the future of the first action, and the third component indicates a final state. Accordingly, $s_0^\triangleright = (s_0^\leq, \emptyset, 0)$, $F^\triangleright = F^\leq \times 2^{Act} \times \{1\}$ and for any $(s, A, f) \in Q^\triangleright$, $a \in Act$, $n \in \{0, \dots, B-1\}$, we set $\delta^\triangleright((s, A, f), (a, n)) =$

$$\begin{cases} (\delta^\leq(s, (a, n)), A \cup \{a\}, 1) & \text{if } \delta^\leq(s, (a, n)) \in F^\leq \text{ and } a \notin A \\ (\delta^\leq(s, (a, n)), A, 0) & \text{otherwise.} \end{cases}$$

Then, the following lemma describes the *nice* property of the above gadgets.

Lemma 5.6. *Let $w = a_1 \dots a_n \in Act^*$ be a B-bounded linearization of an MSC M over Act and ρ be a B-well-stamping for M. Then, for all $1 \leq i \leq j \leq n$, letting $w_{ij}^\rho = (a_i, \rho(i)) \dots (a_j, \rho(j)) \in (Act \times \{0, \dots, B-1\})^*$, we have*

- *$w_{ij}^\rho \in \mathcal{L}(\mathcal{C}^\triangleleft)$ if and only if $(j, i) \in \text{Prev}_{a_i}^M$ and*

- $w^{\rho}_{ij} \in \mathcal{L}(\mathcal{C}^{\triangleright})$ *if and only if* $(i,j) \in \text{Next}^{M}_{a_j}$.

Again, recalling that there is an isomorphism that maps positions of w to events of M, in the above statement $(i,j) \in \text{Prev}^{M}_{a}$ means that the events in M corresponding to the positions i, j are related by Prev^{M}_{a} and so on.

Proof. (1) By definition of $\mathcal{C}^{\triangleleft}$, there is a run r of $\mathcal{C}^{\triangleleft}$ on w^{ρ}_{ij}, namely, $s^{\triangleleft}_0 = s^{\triangleleft}_{i-1} \xrightarrow{(a_i,\rho(i))} s^{\triangleleft}_i \ldots \xrightarrow{(a_j,\rho(j))} s^{\triangleleft}_j$ where for all k, $i-1 \leq k \leq j$, $s^{\triangleleft}_k = (s^{\leq}_k, s'^{\leq}_k, b_k)$. Further note that, $s^{\leq}_{i-1} = s^{\leq}_0$ and $s'^{\leq}_i = s'^{\leq}_{i-1} = s^{\leq}_0$ and for all k, $i \leq k \leq j$, $b_k = a_i$.

Now $s'^{\leq}_j \in F^{\leq}$ implies there is $k, i < k \leq j$ such that $s'^{\leq}_{k+1} \neq s^{\leq}_0$ which means that $a_k = b_k$. But $b_k = a_i$ and so $a_k = a_i$. Also, for this k, $s'^{\leq}_k \xrightarrow{(a_k,\rho(k))} \ldots \xrightarrow{(a_j,\rho(j))} s'^{\leq}_j$ is an accepting run of $\mathcal{C}^{\leq}$ on w^{ρ}_{kj}. Conversely, if $\exists k$, $i < k \leq j$ such that $a_k = a_i$ and there is an accepting run of $\mathcal{C}^{\leq}$ on w^{ρ}_{kj} then in r we find that $s'^{\leq}_j \in F^{\leq}$.

Thus, we can conclude that $s'^{\leq}_j \in F^{\leq}$ if and only if there is $k, i < k \leq j$ such that $a_k = a_i$ and there is an accepting run of $\mathcal{C}^{\leq}$ on w^{ρ}_{kj}. But by Lemma 5.5 there is an accepting run of $\mathcal{C}^{\leq}$ on w^{ρ}_{kj} if and only if $k \leq^{M} j$. Thus $s'^{\leq}_j \in F^{\leq}$ if and only if there is $k, i < k \leq j$ such that $a_k = a_i$ and $k \leq^{M} j$. Also by Lemma 5.5 we have $s^{\leq}_j \in F^{\leq}$ if and only if $i \leq^{M} j$.

Finally r is an accepting run of $\mathcal{C}^{\triangleright}$ on w^{ρ}_{ij} if and only if $s^{\triangleleft}_j \in F^{\triangleleft}$, i.e, $s^{\leq}_j \in F^{\leq}$ and $s'^{\leq}_j \notin F^{\leq}$. And by the above arguments, this happens if and only if $i \leq^{M} j$ and for all k, $i < k \leq j$, either $a_k \neq a_i$ or $k \not\leq^{M} j$. But this is exactly the definition of $(j,i) \in \text{Prev}^{M}_{a_i}$. Thus $w^{\rho}_{ij} \in \mathcal{L}(\mathcal{C}^{\triangleleft})$ if and only if $(i,j) \in \text{Prev}^{M}_{a_i}$.

(2) Similarly, there is a run r' of $\mathcal{C}^{\triangleright}$ on w^{ρ}_{ij}, namely, $s^{\triangleright}_0 = s^{\triangleright}_{i-1} \xrightarrow{(a_i,\rho(i))} s^{\triangleright}_i \ldots \xrightarrow{(a_j,\rho(j))} s^{\triangleright}_j$ where for all k, $i-1 \leq k \leq j$, $s^{\triangleright}_k = (s^{\leq}_k, A_k, b_k)$ for $b_k \in \{0,1\}$, such that $r'_1 = s^{\leq}_{i-1} \xrightarrow{(a_i,\rho(i))} \ldots \xrightarrow{(a_j,\rho(j))} s^{\leq}_j$ is a run of $\mathcal{C}^{\leq}$ on w^{ρ}_{ij}. Then r' is accepting if and only if $s^{\leq}_j \in F^{\leq}$ and $b_j = 1$ i.e, if and only if r'_1 is accepting and $a_j \notin A_{j-1}$. By Lemma 5.5, r'_1 is accepting if and only if $i \leq^{M} j$. Also $a_j \notin A_{j-1}$ implies $a_j \notin A_k$ for all $i \leq k < j$ which means that for all k, $i \leq k < j$, $a_k \neq a_j$. Thus, we conclude that r' is accepting if and only if $i \leq^{M} j$ and for all k, $i \leq k < j$, $a_k \neq a_j$ i.e, if and only if $(i,j) \in \text{Next}^{M}_{a_j}$.

□

5.2 From ECMPA to timed automata

If we ignore the clock constraints and all the timing related issues, then this simulation is exactly the same as giving the semantics of an MPA over linearizations of MSCs as described in Section 2.2.3. However, when we include the timing constraints of the ECMPA, then along the run of the TA we need to know, when to

verify these constraints and against which clocks. We maintain this information in the state space using the gadgets.

Intuitively, along the run of the TA, we start a new copy of the Prev gadget and reset a corresponding clock z. Thus, at a later position, if we encounter a Prev_a constraint for some $a \in Act$, and if *this* copy of the Prev gadget is in a final state, then we know that these positions are related by the Prev_a relation and hence the constraint must be checked against the clock z.

Similarly, at each position of the run where we encounter a Next_a constraint, we start a new copy of the Next gadget and reset a clock z'. Then, when it reaches an accepting state and the last transition is an a-action, we know that these positions are related by Next_a relation. Hence, at this point we verify that clock z' satisfies the constraint mentioned when the gadget was started. However, note that for this, we also need to maintain, in the state space, the constraint itself, so that we can recover it and verify it at the latter position.

Finally, for message constraints, we reset a clock when we encounter the send action and verify it when we reach the correct receive. This information is already contained in the state space as we will see. In addition however, we need to maintain the message constraint itself in the state space, so that when we reach the receive we know which constraint to check the clock against.

Now, we formalize these ideas below. For the rest of this chapter, we fix an ECMPA $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, Act, \Delta, F)$, with $\mathcal{A}_p = (S_p, \iota_p, \to_p)$, and an integer $B > 0$. We also fix an (infinite) set of indices $\text{Ind} = Act \times \mathbb{N}$ which will be used to index the "copies" of the gadgets that we will use. We then define the timed automaton $\mathcal{B} = (Q_\mathcal{B}, \mathcal{Z}_\mathcal{B}, \delta_\mathcal{B}, \iota_\mathcal{B}, F_\mathcal{B})$ as follows. A state $\text{st} \in Q_\mathcal{B}$ is a 6-tuple $(\overline{s}, \chi, \eta, \xi^\triangleleft, \xi^\triangleright, \gamma)$ where:

- $\overline{s} = (s_p)_{p \in Proc} \in \prod_{p \in Proc} S_p$ is a tuple of local states from the ECMPA.
- $\chi : Ch \to (\mathcal{M} \times \Delta)^{\leq B}$ describes the contents of the channels.
- $\eta : Act \to \{0, \ldots, B-1\}$ gives the B-stamping number that should be assigned to the next occurrence of an action.
- $\xi^\triangleleft : \text{Ind} \dashrightarrow Q^\triangleleft$ associates with certain indices, states of $\mathcal{C}^\triangleleft$. Thus, the indices in $\text{dom}(\xi^\triangleleft)$ of a state specify which copies of the gadgets are "active" in that state.
- $\xi^\triangleright : \text{Ind} \dashrightarrow Q^\triangleright \times \text{Int}(\mathcal{A})$ associates with some indices, states of $\mathcal{C}^\triangleright$ along with constraints that need to be maintained
- $\gamma : Ch \times \{0, \ldots, B-1\} \dashrightarrow \text{Int}(\mathcal{A})$ describes the guards attached to messages.

The initial state is $\iota_\mathcal{B} = ((\iota_p)_{p \in Proc}, \chi_0, \eta_0, \xi_0^\triangleleft, \xi_0^\triangleright, \gamma_0)$ where χ_0 and η_0 map any argument to the empty word and 0 respectively, and the partial maps $\xi_0^\triangleleft$, $\xi_0^\triangleright$ and γ_0 are nowhere defined.

We will use clocks from the set $\mathcal{Z}_\mathcal{B} = \{z^{\triangleleft}_{a,i}, z^{\triangleright}_{a,i} \mid (a,i) \in \text{Ind}\} \cup \{z^{\gamma}_{p,q,i} \mid (p,q) \in Ch, i \in \{0,\ldots,B-1\}\}$. Also, we fix some notations regarding how we write constraints using these clocks. For a clock $z \in \mathcal{Z}_\mathcal{B}$ and an interval $I \in \mathcal{I}$ with endpoints $l, r \in \mathbb{N}$, we write "$x \in I$" to denote a constraint from $\text{Form}(\mathcal{Z}_\mathcal{B})$. More specifically, $x \in I$ denotes $x > l \wedge x < r$, if $I = (l,r)$. If $I = [l,r]$, then $x \in I$ refers to $(x > l \vee x = l) \wedge (x < r \vee x = r)$, the constraint $(x > l \vee x = l) \wedge x < r$, if $I = [l,r)$ and the constraint $x > l \wedge (x < r \vee x = r)$, if $I = (l,r]$.

The transition relation $\delta_\mathcal{B} \subseteq Q_\mathcal{B} \times \text{Form}(\mathcal{Z}_\mathcal{B}) \times Act \times 2^{\mathcal{Z}} \times Q_\mathcal{B}$ is defined by,

$$((\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma), \varphi, a, R, (\overline{s}', \chi', \eta', \xi'^{\triangleleft}, \xi'^{\triangleright}, \gamma')) \in \delta_\mathcal{B}$$

if there are $p, q \in Proc, m \in \mathcal{M}, \theta \in \{!, ?\}$ such that $a = p\theta q(m)$ and there exists a p-local transition of the ECMPA, $(s_p, a, g, d, s'_p) \in \rightarrow_p$ for some $g \in [\text{TC} \dashrightarrow \mathcal{I}]$ and $d \in \Delta$ such that the following conditions (i.– ix.) hold:

i. $s'_r = s_r$ for all $r \in Proc \setminus \{p\}$.

ii. if $\theta = !$, then $\chi'(p,q) = (m,d) \cdot \chi(p,q)$ and $\chi'(r,s) = \chi(r,s)$ for all $(r,s) \in Ch \setminus \{(p,q)\}$.

iii. if $\theta = ?$, then $\chi(q,p) = \chi'(q,p) \cdot (m,d)$ and $\chi'(r,s) = \chi(r,s)$ for all $(r,s) \in Ch \setminus \{(q,p)\}$.

iv. for all $b \in Act$,

$$\eta'(b) = \begin{cases} (\eta(b)+1) \bmod B & \text{if } b \in p\theta q \\ \eta(b) & \text{otherwise} \end{cases}$$

v. The states of the *previous* automata are updated and we have initialized a new copy of $\mathcal{C}^{\triangleleft}$ starting at the current position so that we can determine which latter positions are related with the current one by the *previous* relation.

$$\xi'^{\triangleleft}(b,i) = \begin{cases} \delta^{\triangleleft}(s^{\triangleleft}_0, (a, \eta(a))) & \text{if } b = a \text{ and } i = \min(\mathbb{N} \setminus \text{dom}(\xi^{\triangleleft}(a))) \\ \delta^{\triangleleft}(\xi^{\triangleleft}(b,i), (a, \eta(a))) & \text{if } (b,i) \in \text{dom}(\xi^{\triangleleft}) \\ \text{undefined} & \text{otherwise} \end{cases}$$

vi. The states of the *next* automata are updated along with the corresponding guards. A new copy of $\mathcal{C}^{\triangleright}$ is initialized for each $b \in Act$, if there is a Next_b constraint on the local transition. The guard itself is stored in the second component, so that it can be verified when we reach the *next* occurence of the action. Once verified, we release the guard and the corresponding copy

of $\mathcal{C}^{\triangleright}$.

$$\xi'^{\triangleright}(b,i) = \begin{cases} (\delta^{\triangleright}(s_0^{\triangleright},(a,\eta(a))), g(\mathrm{Next}_b)) & \text{if } \mathrm{Next}_b \in \mathrm{dom}(g) \text{ and} \\ & \quad i = \min(\mathbb{N} \setminus \mathrm{dom}(\xi^{\triangleright}(b))) \\ (\delta^{\triangleright}(s^{\triangleright},(a,\eta(a))), I) & \text{if } \xi^{\triangleright}(b,i) = (s^{\triangleright}, I) \text{ and} \\ & \quad \neg\big(a = b \wedge \delta^{\triangleright}(s^{\triangleright},(a,\eta(a))) \in F^{\triangleright}\big) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

vii. The guards attached to message constraints are maintained as expected. A send event introduces a constraint, which is retained till its matching receive releases it.

$$\gamma'((r,s),i) = \begin{cases} g(\mathrm{Msg}) & \text{if } a \in r!s, i = \eta(a), \mathrm{Msg} \in \mathrm{dom}(g) \\ \text{undefined} & \text{if } a \in s?r \text{ and } i = \eta(a) \\ \gamma((r,s),i) & \text{otherwise.} \end{cases}$$

viii. A clock is reset for every new copy of $\mathcal{C}^{\triangleleft}$, $\mathcal{C}^{\triangleright}$ and message constraint introduced at this transition.

$$R = \{z^{\triangleleft}_{a,i} \mid i = \min(\mathbb{N} \setminus \mathrm{dom}(\xi^{\triangleleft}(a)))\} \cup \{z^{\gamma}_{p,q,i} \mid a \in p!q, i = \eta(a), \mathrm{Msg} \in \mathrm{dom}(g)\} \\ \cup \{z^{\triangleright}_{b,i} \mid \mathrm{Next}_b \in \mathrm{dom}(g) \text{ and } i = \min(\mathbb{N} \setminus \mathrm{dom}(\xi^{\triangleright}(b)))\}.$$

ix. The guard must ensure that all constraints that get *matched* at the current event are satisfied. Thus $\varphi = \varphi^{\triangleleft} \wedge \varphi^{\triangleright} \wedge \varphi^{m}$ where,

$$\varphi^{\triangleleft} = \bigwedge_{\substack{\{(b,i) \mid \mathrm{Prev}_b \in \mathrm{dom}(g) \\ \text{and } \xi'^{\triangleleft}(b,i) \in F^{\triangleleft}\}}} z^{\triangleleft}_{b,i} \in g(\mathrm{Prev}_b) \quad \wedge \quad \bigwedge_{\substack{\{b \mid \mathrm{Prev}_b \in \mathrm{dom}(g) \\ \text{and } \{i \mid \xi'^{\triangleleft}(b,i) \in F^{\triangleleft}\} = \emptyset\}}} \text{false}$$

ensures that all previous constraints that are matched are satisfied. Thus if the local transition contains a Prev_b constraint, then we have to check $z^{\triangleleft}_{b,i} \in g(\mathrm{Prev}_b)$ for the (unique) i such that $\xi'^{\triangleleft}(b,i) \in F^{\triangleleft}$. If there is no such i then there is no b-action in the past of the current event and the Prev_b constraint of the local transition cannot be satisfied. In this case, we set $\varphi^{\triangleleft}$ to false. Next,

$$\varphi^{\triangleright} = \bigwedge_{\substack{\{i \in \mathrm{dom}(\xi^{\triangleright}(a)) \mid \xi^{\triangleright}(a,i) = (s^{\triangleright}, I), \\ \delta^{\triangleright}(s^{\triangleright},(a,\eta(a))) \in F^{\triangleright}\}}} z^{\triangleright}_{a,i} \in I$$

is for next constraints. If the current action is the next occurence of a from some positions where a *next* guard was registered, for each there is a copy

(a, i) of $\mathcal{C}^{\triangleright}$ which reaches a final state. Thus, we verify the corresponding clock with the constraint recovered from $\xi^{\triangleright}$. For message constraints,

$$\varphi^m = \bigwedge_{\substack{\{((q,p),i) \in \mathrm{dom}(\gamma) \mid \\ a \in p?q,\ \eta(a)=i\}}} z^{\gamma}_{q,p,i} \in \gamma((q,p),i)$$

Finally, the set of final states is $F_{\mathcal{B}} = \{(\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma) \in Q_{\mathcal{B}} \mid \overline{s} \in F, \chi = \chi_0, \mathrm{dom}(\xi^{\triangleright}) = \mathrm{dom}(\gamma) = \emptyset\}$. This ensures that each registered guard has been checked. Indeed, a *next* or *message* constraint is released only when it is checked with the guard φ.

One critical observation here is that, once we have specified the local transition of $\mathcal{A}$, this global transition of $\mathcal{B}$ gets determined uniquely. Thus, this step is always deterministic. Note that the above automaton $\mathcal{B}$ has no ε-transitions either.

Now, we prove that $\mathcal{L}_{tw}(\mathcal{B})$ contains precisely the B-bounded timed linearizations of $\mathcal{L}_{time}(\mathcal{A})$. That is,

Theorem 5.7. $\mathcal{L}_{tw}(\mathcal{B}) = \{\sigma \in (Act \times \mathbb{R}^{\geq 0})^* \mid$ *there is* $T \in \mathcal{L}_{time}(\mathcal{A})$ *such that* σ *is a* B*-bounded timed linearization of* $T\}$.

Proof. Let $T = (M, t)$ with $M = (E, \leq, \lambda)$ over Act. Then, a B-bounded timed linearization $\sigma = (a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$ of T generates the corresponding B-bounded linearization of M, namely $a_1 \ldots a_n$ over the same set of positions $\{1, \ldots, n\}$. Thus, as stated in the previous section, we can interpret the events from E to be positions from $\{1, \ldots, n\}$.

Then recall that $T \in \mathcal{L}_{time}(\mathcal{A})$ if and only if there is an accepting run r of $\mathcal{A}$ on T, where $r : E \to \bigcup_{p \in Proc} S_p$ is given by Definition 3.13. Also recall that $\sigma \in \mathcal{L}_{tw}(\mathcal{B})$ if and only if there is an accepting run r' of $\mathcal{B}$ on σ, i.e, by Definition 2.20, there is a sequence,

$$r' = (\mathrm{st}_0, \nu_0) \xrightarrow{a_1, t_1} (\mathrm{st}_1, \nu_1) \xrightarrow{a_2, t_2} \ldots \xrightarrow{a_n, t_n} (\mathrm{st}_n, \nu_n) \tag{5.3}$$

where for all $i \in \{0, \ldots, n\}$, $\mathrm{st}_i = (\overline{s}_i, \chi_i, \eta_i, \xi^{\triangleleft}_i, \xi^{\triangleright}_i, \gamma_i) \in Q_{\mathcal{B}}$ (with $\overline{s}_i = (s^i_p)_{p \in Proc} \in \prod_{p \in Proc} S_p$), $\mathrm{st}_0 = \iota_{\mathcal{B}}$, $\mathrm{st}_n \in F_{\mathcal{B}}$ and $\nu_i : \mathcal{Z}_{\mathcal{B}} \to \mathbb{R}_{\geq 0}$ (with $\nu_0(x) = 0, \forall x \in \mathcal{Z}_{\mathcal{B}}$) and for each $i \in \{1, \ldots, n\}$, there exist $\varphi_i \in Form(\mathcal{Z}_{\mathcal{B}})$, $R_i \in 2^{\mathcal{Z}_{\mathcal{B}}}$ such that,

$$(\mathrm{st}_{i-1}, \varphi_i, a_i, R_i, \mathrm{st}_i) \in \delta_{\mathcal{B}} \tag{5.4}$$

$$(\nu_{i-1} + t_i - t_{i-1}) \models \varphi_i, \tag{5.5}$$

$$\nu_i = (\nu_{i-1} + t_i - t_{i-1})[R_i \mapsto 0] \tag{5.6}$$

We now show that from an accepting r of $\mathcal{A}$ on T, we can construct an accepting run r' of $\mathcal{B}$ on σ and vice versa.

($\Longrightarrow$) Let an accepting run r of $\mathcal{A}$ on T. Then, we construct run r' inductively from $i = 0$ to $i = n$. Further, at each step, we will maintain two further state invariants. That is, for all $i \in \{0, \ldots, n\}$ we require

$$\forall p \in \mathit{Proc}, s_p^i = \begin{cases} r(j) & \text{if } \exists j \leq i, j \in E_p, \nexists k \in E_p, j < k \leq i. \\ \iota_p & \text{otherwise} \end{cases} \tag{5.7}$$

$$\forall b \in \mathit{Act}, \eta_i(b) = |\{i' \leq i \mid \lambda(i') \in p\theta q\}| \bmod B, \text{ for } b = p\theta q(m) \tag{5.8}$$

Then, indeed at $i = 0$, we have $\mathrm{st}_0 = \iota_{\mathcal{B}} = (\overline{s}_0, \chi_0, \eta_0, \xi_0^{\triangleleft}, \xi_0^{\triangleright}, \gamma_0)$. This satisfies our state invariants (5.7-5.8) since $s_p^0 = \iota_p$ for all $p \in \mathit{Proc}$ and $\eta_0(a) = 0$ for all $a \in \mathit{Act}$. Now for some $i \in \{1, \ldots, n\}$ assuming we have constructed the run r' till $(\mathrm{st}_{i-1}, \nu_{i-1})$, let $a_i = p\theta q(m)$ for some $p, q \in \mathit{Proc}, m \in \mathcal{M}$ and $\theta \in \{!, ?\}$. We then extend the run r' to stage i by exhibiting st_i, ν_i, φ_i and R_i such that Conditions (5.4–5.8) hold.

From the definition of r, we have a local transition on the ECMPA on event i. More precisely, $(r^-(i), a_i, g_i, d_i, r(i)) \in \rightarrow_p$ for some guard g_i and $d_i \in \Delta$. Recall that $r^-(i) = r(i')$ for $i' \lessdot_{pp} i$ and $r^-(i) = \iota_p$ if such an event i' does not exist. Thus, by Condition (5.7) at stage $i-1$, we have $s_p^{i-1} = r^-(i)$. And by choosing $s_p^i = r(i)$ we obtain $(s_p^{i-1}, a_i, g_i, d_i, s_p^i) \in \rightarrow_p$. Again, by choosing for all $p' \neq p$, $s_{p'}^i = s_{p'}^{i-1}$, Condition (5.7) holds at stage i. (This follows since for p, the largest $j \leq i$ such that $j \in E_p$ is i itself. And for $p' \neq p$, the largest $j' \leq i$ such that $j' \in E_{p'}$ is the largest such event $j' \leq i-1$).

Now, as we commented in our construction the local transition fully specifies the global transition and thus we get a transition of $\mathcal{B}$,

$$((\overline{s}_{i-1}, \chi_{i-1}, \eta_{i-1}, \xi_{i-1}^{\triangleleft}, \xi_{i-1}^{\triangleright}, \gamma_{i-1}), \varphi_i, a_i, R_i, (\overline{s}_i, \chi_i, \eta_i, \xi_i^{\triangleleft}, \xi_i^{\triangleright}, \gamma_i)) \in \delta_{\mathcal{B}}$$

where $\varphi_i, R_i, \chi_i, \eta_i, \xi_i^{\triangleleft}, \xi_i^{\triangleright}, \gamma_i$ are defined from their values at stage $i-1$ and the local transition. Thus Condition (5.4) holds at i, with $\mathrm{st}_i = (\overline{s}_i, \chi_i, \eta_i, \xi_i^{\triangleleft}, \xi_i^{\triangleright}, \gamma_i)$.

Again Condition (5.8) continues to hold at i if it holds at $i-1$. (If $b \in p\theta q$, then $\eta_i(b) = (\eta_{i-1}(b) + 1) \bmod B = |\{i' \leq i-1 \mid \lambda(i') \in p\theta q\}| + 1 \mod B = |\{i' \leq i \mid \lambda(i') \in p\theta q\}| \mod B$. And for $b \in \mathit{Act} \setminus p\theta q$, it follows since $\eta_i(b) = \eta_{i-1}(b)$.) Now, we just define $\nu_i = (\nu_{i-1} + t_i - t_{i-1})[R_i \mapsto 0]$, so that Condition (5.6) holds at i. Thus, we have extended run r' of $\mathcal{B}$ on σ to i, if we prove Condition (5.5), i.e,

Claim 5.8. $(\nu_{i-1} + t_i - t_{i-1}) \models \varphi_i$

Proof. The proof is by induction on the structure of φ_i. But first we observe that the map $\rho : E \to \{0, \ldots, B-1\}$ which maps $\rho(i) = \eta_i(a_i)$ is a B-well-stamping for M. This follows from the fact that the state invariant, Condition (5.8), holds till stage i. We have the following cases to consider.

(1) *Previous* constraint of the form $z_{a,k}^{\triangleleft} \in g_i(\mathrm{Prev}_a)$ or false: If for some $a \in \mathit{Act}$, $\mathrm{Prev}_a \in \mathrm{dom}(g_i)$, then by definition of run r of $\mathcal{A}$ on T, (i.e, Condition (3.5) or

(3.6)), there exists event j, $(i,j) \in \mathrm{Prev}_a^T$ (thus, $a_j = a$). Thus by Lemma 5.6, $(a_j, \eta_j(a_j)) \ldots (a_i, \eta_i(a_i)) \in \mathcal{L}(\mathcal{C}^\triangleleft)$. And so, in the run of $\mathcal{B}$ at stage i, $\xi_i^\triangleleft(a,k) \in F^\triangleleft$ where $k = \min(\mathbb{N} \setminus \mathrm{dom}(\xi_{j-1}^\triangleleft(a)))$. Hence for any a such that $\mathrm{Prev}_a \in \mathrm{dom}(g_i)$, the set $\{\ell \mid \xi'^\triangleleft(a,\ell) \in F^\triangleleft\} \neq \emptyset$ and so the *false* constraint cannot occur as a guard in this simulation, i.e, as part of φ_i.

Now, at stage j, $z_{a,k}^\triangleleft \in R_j$ (there cannot be another Prev_a guard for some other k' since there is a unique preceding occurrence of each letter). Also $z_{a,k}^\triangleleft \notin R'_j$ for all $j' \in \{j+1, \ldots, i\}$, since $(a,j') \in \mathrm{dom}(\xi_{j'-1}^\triangleleft)$. Therefore in the valuation $(\nu_{i-1} + t_i - t_{i-1})(z_{a,k}^\triangleleft) = \nu_{i-1}(z_{a,k}^\triangleleft) + t_i - t_{i-1} = \nu_{i-2}(z_{a,k}^\triangleleft) + t_{i-1} - t_{i-2} + t_i - t_{i-1} = \ldots = \nu_j(z_{a,k}^\triangleleft) + t_i - t_j$ and $\nu_j(z_{a,k}^\triangleleft) = 0$. Thus $\nu_j(z_{a,k}^\triangleleft) + t_i - t_j = t_i - t_j$ and so we are done if we show that $t_i - t_j \in g_i(\mathrm{Prev}_a)$. But this follows from Condition (3.5) (or 3.6), where we have $|t(j) - t(i)| \in g_i(\mathrm{Prev}_a)$ where $t(j) = t_j$ and $t(i) = t_i$. Thus we conclude that the valuation satisfies any *previous* clock constraint of the form $z_{a,k}^\triangleleft \in g_i(\mathrm{Prev}_a)$.

(2) *Next* constraint of the form $z_{a_i,k}^\triangleright \in I$: This implies that there exists $k \in \mathrm{dom}(\xi_{i-1}^\triangleright(a_i))$ such that, $\xi_{i-1}^\triangleright(a_i,k) = (s^\triangleright, I)$ and $\delta^\triangleright(s^\triangleright, (a_i, \eta_i(a_i))) \in F^\triangleright$. Then, by the definition of the next update function $\xi^\triangleright$, we can conclude that there exists $j < i$, $k = \min(\mathbb{N} \setminus \mathrm{dom}(\xi_{j-1}^\triangleright(a_i)))$ such that $\xi_j^\triangleright(a_i,k) = (\delta^\triangleright(s_0^\triangleright, (a_j, \eta(a_j)), I))$ such that $I = g_j(\mathrm{Next}_{a_i})$. Thus, $(a_j, \eta_j(a_j)) \ldots (a_i, \eta_i(a_i)) \in L(\mathcal{C}^\triangleright)$ and by Lemma 5.6, we can conclude that $(j,i) \in \mathrm{Next}_{a_i}^T$. Hence from the definition of run r on T-MSC T, we get $|t(j) - t(i)| \in g_j(\mathrm{Next}_{a_i}) = I$. Also, $z_{a_i,k}^\triangleright \in R_j$ and it is not reset till i. If not, let j', $j < j' < i$ be the first instance where, $z_{a_i,k}^\triangleright \in R_{j'}$. This $(a_i,k) \notin \mathrm{dom}(\xi_{j'}^\triangleright)$ and $(a_i,k) \in \mathrm{dom}(\xi_{j'-1}^\triangleright)$ implies that $a_{j'} = a_i$ and $(a_j, \eta(j)) \ldots (a_{j'}, \eta(j')) \in \mathcal{L}(\mathcal{C}^\triangleright)$ which contradicts $(j,i) \in \mathrm{Next}_{a_i}$. Thus, for all $j < j' < i$ we get $z_{a_i,k}^\triangleright \notin R_{j'}$. Thus $(v_{i-1} + t_i - t_{i-1})(z_{a_i,k}^\triangleright) = t_i - t_j \in I$ and so we are done.

(3) *Message* constraint of the form $z_{q,p,k}^\gamma \in \gamma_{i-1}(q,p,k)$: Here, $(q,p,k) \in \mathrm{dom}(\gamma_{i-1})$ such that $a_i \in p?q$, $\eta_{i-1}(a_i) = k$. Then we look at the largest $j \leq i$ such that $a_j = q!p$ and $\eta_{j-1}(a_j) = k$. Such a j exists since if not it would contradict the fact that σ is a linearization of a valid T-MSC T. Further for all j', $j < j' < i$, it is not the case that $a_{j'} \in p?q$ and $\eta_{j'-1}(a_{j'}) = k$. Then it follows that for all j', $j \leq j' < i$, $\gamma_{j'}(q,p,k) = g_j(\mathrm{Msg})$. Also $(j,i) \in \mathrm{Msg}^T$ and so by definition of r on TMSC, $|t_j - t_i| \in g_j(\mathrm{Msg}) = \gamma_{i-1}(q,p,k)$. And again, $z_{q,p,k}^\gamma \in R_j$ and for all $j < j' \leq i$, $z_{q,p,k}^\gamma \notin R_{j'}$, so $(v_{i-1} + t_i - t_{i-1})(z_{q,p,k}^\gamma) = t_i - t_j \in \gamma_{i-1}(q,p,k)$ and thus we are done. □

Now it is easy to see that the state $\widetilde{s}_n = (\overline{s}_n, \chi_n, \eta_n, \xi_n^\triangleleft, \xi_n^\triangleright, \gamma_n)$ reached at the end of the above run, is a final state. This follows from the fact that r is a successful run of $\mathcal{A}$ on T, since then we have, $\overline{s}_n \in F$ and $\chi_n = \chi_0$ (since at the end of r the channel contents must be empty), and the partial maps $\xi^\triangleright$ and γ are nowhere defined (since if that were not the case then this means that a constraint was not checked with its guard).

($\Longleftarrow$) For the converse, from r' as defined in (5.3) above, we want to con-

struct r a run of $\mathcal{A}$ on T, i.e, a map $r : E \to \bigcup_{p\in Proc} S_p$. We define for each event $i \in \{1,\ldots,n\}$, $r(i) = s^i_{p_i}$. Now, at each $i \in \{0,\ldots,n\}$, by (5.4), we have $(\mathrm{st}_{i-1}, a_i, \varphi_i, R_i, \mathrm{st}_i) \in \delta_{\mathcal{B}}$. By definition of $\mathcal{B}$, for each $i \in \{0,\ldots,n\}$, we have $(s^{i-1}_{p_i}, a_i, g_i, d_i, s^i_{p_i}) \in \to_{p_i}$ for some g_i, d_i. We now show that this map is a run of $\mathcal{A}$ on T by verifying Conditions (3.2,3.5-3.6) in Definition 3.13.

First observe that $r^-(i) = s^{i-1}_{p_i}$. This can be proved by induction on i. For $i = 1$, $\mathrm{st}_0 = \iota_{\mathcal{B}}$ implies that $s^0_{p_1} = \iota_{p_1}$ and $r^-(a_1) = \iota_{p_1}$ since a_1 is the minimal event in that process. For $i > 1$, $r^-(i) = r(j)$ if there exists $j \lessdot_{p_ip_i} i$ and $r^-(i) = \iota$ otherwise. Thus we have $s^{i-1}_{p_i} = s^j_{p_j}$.

Now, for events i, j, suppose $i <^M_{pq} j$ then $(r^-(i), a_i, g_i, d_i, r(i)) \in \to_{p_i}$ and $(r^-(j), a_j, g_j, d_j, r(j)) \in \to_{p_j}$ implies that $d_i = d_j$ which means that Condition (3.2) holds. This follows from definition of χ in $\delta_{\mathcal{B}}$ which ensures that the sent and received messages are synchronized. To prove the other conditions, for any event i, and $\alpha \in \mathrm{TC}$, we need to show that, if $\alpha \in \mathrm{dom}(g_i)$, then $\exists j, (i,j) \in \mathrm{dom}(\alpha^T)$ such that $|t(i) - t(j)| \in g_i(\alpha)$. We have three cases depending on α.

- $\alpha = \mathrm{Prev}_a$ for some $a \in Act$. If $\mathrm{Prev}_a \in \mathrm{dom}(g_i)$, then firstly there exists some (unique) k such that $\xi^{\triangleleft}_i(a,k) \in F^{\triangleleft}$. If not, then the *false* constraint will occur in φ_i which contradicts acceptance of σ by $\mathcal{B}$. Thus we have that for this (a,k), the constraint $z^{\triangleleft}_{a,k} \in g_i(\mathrm{Prev}_a)$ occurs in φ_i and $(\nu_{i-1}+t_i-t_{i-1})(z^{\triangleleft}_{a,k}) \models \varphi_i$ implies that $(\nu_{i-1} + t_i - t_{i-1})(z^{\triangleleft}_{a,k}) \in g_i(\mathrm{Prev}_a)$.

  Now by definition of *previous* state updates in the run of $\mathcal{B}$, we can use Lemma 5.6 to conclude that there exists $j \leq i$ for which $(i,j) \in \mathrm{Prev}^M_a$. Further at j we can see that, $z^{\triangleleft}_{a,k} \in R_j$ and for all j',$j < j' < i$, $z^{\triangleleft}_{a,k} \notin R_{j'}$. Thus $(\nu_{i-1} + t_i - t_{i-1})(z^{\triangleleft}_{a,k}) = t_i - t_j$. Hence we conclude that $(i,j) \in \mathrm{Prev}^T_a$ and $|t(i) - t(j)| \in g_i(\mathrm{Prev}_a)$.

- $\alpha = \mathrm{Next}_a$ for some $a \in Act$. If $\mathrm{Next}_a \in \mathrm{dom}(g_i)$, then by the definition of $\xi^{\triangleright}$, for $k = \min(\mathbb{N}\backslash\mathrm{dom}(\xi^{\triangleright}_{i-1}(a)))$, we have $\xi^{\triangleright}_i(a,k) = (\delta^{\triangleright}(s^{\triangleright}_0, (a_i, \eta(a_i)), g_i(\mathrm{Next}_a))$. Also we reset the clock $z^{\triangleright}_{a,k}$

  But since r' is an accepting run, $\xi^{\triangleright}_n(a,k)$ is undefined and so there exists j such that $a = a_j$ and $\xi^{\triangleright}_j(a,k) = (s, I)$ with $s \in F^{\triangleright}$. Let j be the smallest such j so that for j', $i < j' < j$, $(a,k) \in \mathrm{dom}(\xi^{\triangleright}_{j'})$. Thus, we note that $I = g_i(\mathrm{Next}_a)$. This also ensures that $(a_i, \eta_i(a_i))\ldots(a_j, \eta_j(a_j)) \in \mathcal{L}(\mathcal{C}^{\triangleright})$ and so by Lemma 5.6, $(i,j)\mathrm{Next}^T_a$. Now, we have $(\nu_{j-1} + t_j - t_{j-1})(z^{\triangleright}_{a,k}) = \nu_i(z^{\triangleright}_{a,k}) + t_j - t_i = t_j - t_i$. Again because of the successful run, $z^{\triangleright}_{a,k} \in I$ occurs in φ_j and $(\upsilon_{j-1} + t_j - t_{j-1})(z^{\triangleright}_{a,k}) \in I$. This implies from above that $t_j - t_i \in g_i(\mathrm{Next}_\sigma)$ and so we are done.

- $\alpha = \mathrm{Msg}$. This is similar to above case of *next* (and in fact simpler since it does not need the *next* automata construction).

This completes both directions of the proof.

□

5.3 A finite version of $\mathcal{B}$

To get a finite version of $\mathcal{B}$, we will bound the set of indices Ind to a finite set Ind_{fin}, thus, constructing a finite timed automaton $\mathcal{B}'$ that is equivalent to $\mathcal{B}$. The state space of $\mathcal{B}'$ is the same as for $\mathcal{B}$ except that it uses indices from Ind_{fin} to define the $\xi^{\triangleleft}$ and $\xi^{\triangleright}$ components. We will construct $\mathcal{B}'$ in two steps. First, we describe how the transitions of $\mathcal{B}$ are modified to handle the previous gadgets. Next, we describe how to modify the transitions of $\mathcal{B}$ to handle the next gadgets. Thus, $\mathcal{B}'$, the timed automaton obtained after both these modifications are made will turn out to be a finite timed automaton.

To bound the set of indices, our basic idea is to reuse copies of the previous and next gadgets when it is safe to do so. First, we handle the *previous* case by examining when it is "safe" to release a copy of $\mathcal{C}^{\triangleleft}$. The following proposition gives us the criterion required.

Proposition 5.9. *Let $(\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma)$ be a reachable state of $\mathcal{B}$. If there exist two indices $(a,i),(a,j) \in \text{dom}(\xi^{\triangleleft}), i \neq j$ such that $\xi^{\triangleleft}(a,i) = \xi^{\triangleleft}(a,j) = s^{\triangleleft} \in Q^{\triangleleft}$, then no final state of $s_f^{\triangleleft} \in F^{\triangleleft}$ is reachable from $s^{\triangleleft}$.*

Proof. The copies of $\mathcal{C}^{\triangleleft}$ indexed by (a,i) and (a,j) have been started at distinct positions labeled a to keep track of two different pasts. Now suppose there exists $s_f^{\triangleleft} \in F^{\triangleleft}$, such that $s_f^{\triangleleft}$ is reachable from $s^{\triangleleft}$. Then at $s_f^{\triangleleft}$, this position is related by Prev_a with both starting positions, i.e., when the clocks $z_{a,i}^{\triangleleft}$ and $z_{a,j}^{\triangleleft}$ were last reset. But this is not possible, because there is at most one previous position labeled a for any position. Thus no final state is reachable from $s^{\triangleleft}$. □

This implies that we can safely remove the corresponding indices (a,i) and (a,j) from $\text{dom}(\xi^{\triangleleft})$. Thus, we say that a state $\text{st} = (\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma) \in Q_{\mathcal{B}}$ is $\triangleleft$-*safe* if there are no two indices $(a,i),(a,j) \in \text{dom}(\xi^{\triangleleft}), i \neq j$ such that $\xi^{\triangleleft}(a,i) = \xi^{\triangleleft}(a,j)$. Otherwise, we say that st is $\triangleleft$-*unsafe* and that $(a,i),(a,j)$ are $\triangleleft$-*unsafe* indices at the state st.

A transition from $\mathcal{B}$ is retained in $\mathcal{B}'$ if it is between $\triangleleft$-safe states. In addition, every transition $(\text{st}, \varphi, a, R, \text{st}')$ in $\mathcal{B}$ from a $\triangleleft$-safe state st to a $\triangleleft$-unsafe state $\text{st}' = (\overline{s}', \chi', \eta', \xi'^{\triangleleft}, \xi'^{\triangleright}, \gamma')$ is replaced by a transition $(\text{st}, \varphi, a, R, \widetilde{\text{st}'})$ between $\triangleleft$-safe states, where $\widetilde{\text{st}'} = (\overline{s}', \chi', \eta', \widetilde{\xi'^{\triangleleft}}, \xi'^{\triangleright}, \gamma')$ and,

$$\widetilde{\xi'^{\triangleleft}}(b,i) = \begin{cases} \xi'^{\triangleleft}(b,i) & \text{if } (b,i) \in \text{dom}(\xi'^{\triangleleft}) \text{ and } \nexists j \neq i \text{ such that} \\ & (b,j) \in \text{dom}(\xi'^{\triangleleft}) \text{ and } \xi'^{\triangleleft}(b,i) = \xi'^{\triangleleft}(b,j) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{5.9}$$

By Proposition 5.9, $\mathcal{B}'$ still accepts the same set of timed words as $\mathcal{B}$. The following lemma states that this is enough to ensure finiteness in the *previous* case.

Lemma 5.10. *For any reachable state* $(\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma) \in Q_{\mathcal{B}'}$*, we have* $\mathrm{dom}(\xi^{\triangleleft}) \subseteq Act \times \{0, \ldots, |Q^{\triangleleft}|\}$.

Proof. Suppose not, then for some $a \in Act$, $|\mathrm{dom}(\xi^{\triangleleft}, a)| \geq (|Q^{\triangleleft}| + 1)$. But this implies that must exist at least two indices $(a, i), (a, j) \in \mathrm{dom}(\xi^{\triangleleft}), i \neq j$ such that $\xi^{\triangleleft}(a, i) = \xi^{\triangleleft}(a, j)$. By the above definition, they would have been undefined and hence cannot be in the domain of $\xi^{\triangleleft}$. Thus we have a contradiction. □

The remaining source of infinity comes from *next* constraints. The situation is not as easy as for previous constraints, since next constraints registered at several positions could be matched at the same time. Thus, the number of registered Next_b constraints may be unbounded. In particular in some state of $\mathcal{B}$, suppose $(b, i), (b, j) \in \mathrm{dom}(\xi^{\triangleright})$ for some $i \neq j$ and $\xi^{\triangleright}(b, i) = (s^{\triangleright}, I), \xi^{\triangleright}(b, j) = (s^{\triangleright}, I')$, then the constraints associated with i and j will be matched simultaneously. When matched, the guard on the transition of $\mathcal{B}$ will include both $z^{\triangleright}_{b,i} \in I$ and $z^{\triangleright}_{b,j} \in I'$. The idea is that we need to keep only the *stronger constraint* and can release the other one. To determine the stronger constraint we have to deal separately with the upper parts and the lower parts of the constraints.

Refining the constraints A clock constraint over $\mathcal{Z}_{\mathcal{B}}$ is called an *upper*-guard if it is of the form $x \sim c$ where $\sim \in \{<, \leq\}$ for some $x \in \mathcal{Z}_{\mathcal{B}}, c \in \mathbb{Q}_{\geq 0}$. Similarly $x \sim c$ is a *lower*-guard if $\sim \in \{>, \geq\}$. Note that each $x \in I$ defines uniquely a lower and an upper-guard, depending upon the endpoints of the interval I.

Definition 5.11. *Let* $x \sim c$ *and* $x' \sim' c'$ *be two upper-guards with* $\sim, \sim' \in \{<, \leq\}$ *or two lower guards with* $\sim, \sim' \in \{>, \geq\}$*. We say* $x \sim c$ *is* stronger than $x' \sim' c'$ *if, when evaluated at the same instant,* $x \sim c$ *holds implies* $x' \sim' c'$ *holds as well.*

The stronger constraint can be determined with a diagonal guard: For upper guards, $x \sim c$ is stronger than $x' \sim' c'$ if either $x' - x < c' - c$ or else $x' - x \leq c' - c$ and ($\sim = <$ or $\sim' = \leq$). The relation *stronger than* is transitive among upper-guards. It is also total: either $x \sim c$ is stronger than $x' \sim' c'$, or the converse holds, or both in which case we say that the two constraints are equivalent. Note that the constraints are *equivalent* if and only if $x' - x = c' - c$ and $\sim = \sim'$. The above properties are true for lower guards as well, where we have $x \sim c$ stronger than $x' \sim' c'$ if either $x' - x > c' - c$ or else $x' - x \geq c' - c$ and ($\sim = >$ or $\sim' = \geq$).

Restricting the domain of $\xi^{\triangleright}$ Now we get back to our problem and show how to change $\mathcal{B}$ so that the size of $\mathrm{dom}(\xi^{\triangleright})$ in a state $\mathrm{st} = (\overline{s}, \chi, \eta, \xi^{\triangleleft}, \xi^{\triangleright}, \gamma)$ can be bounded by $|Act| \cdot (2|Q^{\triangleright}| + 1)$. Note that a transition of $\mathcal{B}$ may initiate at most $|Act|$ new copies of $\mathcal{C}^{\triangleright}$ (one for each $b \in Act$ such that $\mathrm{Next}_b \in \mathrm{dom}(g)$). Hence, we say that state st is $\triangleright$-*safe* if for all $b \in Act$ we have $|\mathrm{dom}(\xi^{\triangleright}(b))| \leq 2|Q^{\triangleright}|$. Thus, transitions of $\mathcal{B}$ are retained in $\mathcal{B}'$ only when they are between $\triangleright$-safe states.

For a state st, $b \in Act$, $s^{\triangleright} \in Q^{\triangleright}$, we define $\mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright}) = \{i \mid \xi^{\triangleright}(b, i) = (s^{\triangleright}, I), I \in \mathcal{I}\}$. Also for $i \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$, we denote by I_i the interval such that $\xi^{\triangleright}(b, i) = (s^{\triangleright}, I_i)$. Now, if the state st is not $\triangleright$-safe, then there exist $b \in Act, s^{\triangleright} \in Q^{\triangleright}$ such that $|\mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})| > 2$. In this case, we say that st is $\triangleright$-unsafe for $(b, s^{\triangleright})$.

Now, for each $(b, s^{\triangleright})$ such that st is $\triangleright$-unsafe for $(b, s^{\triangleright})$, we can find $i_\ell, i_u \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$ such that the lower-guard defined by $z^{\triangleright}_{b,i_\ell} \in I_{i_\ell}$ is stronger than all lower-guards defined by $z^{\triangleright}_{b,j} \in I_j$ for $j \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$ and the upper-guard defined by $z^{\triangleright}_{b,i_u} \in I_{i_u}$ is stronger than all upper-guards defined by $z^{\triangleright}_{b,j} \in I_j$ for $j \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$. From the definition of the relation *stronger than*, all constraints $z^{\triangleright}_{b,j} \in I_j$ for $j \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$ are subsumed by the conjunction of $z^{\triangleright}_{b,i_\ell} \in I_{i_\ell}$ and $z^{\triangleright}_{b,i_u} \in I_{i_u}$. Therefore, we can release all next-constraints associated with (b, j) with $j \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright}) \setminus \{i_\ell, i_u\}$.

To do this in the automaton, we define guards of the form $\varphi(i_\ell^{b,s^{\triangleright}}, i_u^{b,s^{\triangleright}})$ that evaluate to true if $i_\ell^{b,s^{\triangleright}}$ and $i_u^{b,s^{\triangleright}}$ determine stronger lower- and upper-constraints among those defined by $\mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$. Since the relation *stronger than* can be expressed with diagonal constraints as we have seen above, we have $\varphi(i_\ell^{b,s^{\triangleright}}, i_u^{b,s^{\triangleright}}) \in \mathrm{Form}(\mathcal{Z}_{\mathcal{B}})$.

Thus, for each transition $(\mathrm{st}, \varphi, a, R, \mathrm{st}')$ in $\mathcal{B}$ from a $\triangleright$-safe state st to a state $\mathrm{st}' = (\overline{s}', \chi', \eta', \xi'^{\triangleleft}, \xi'^{\triangleright}, \gamma')$ that is not $\triangleright$-safe, we replace it by a transition $(\mathrm{st}, \varphi', a, R, \widetilde{\mathrm{st}'})$, where $\varphi' = \varphi \wedge \Big(\bigwedge_{|\mathrm{Activ}_{\mathrm{st}'}(b,s^{\triangleright})|>2} \varphi(i_\ell^{b,s^{\triangleright}}, i_u^{b,s^{\triangleright}}) \Big)$ and $\widetilde{\mathrm{st}'} = (\overline{s}', \chi', \eta', \xi'^{\triangleleft}, \widetilde{\xi'^{\triangleright}}, \gamma')$ is such that,

$$\widetilde{\xi'^{\triangleright}}(b, i) = \begin{cases} \xi'^{\triangleright}(b, i) & \text{if } \exists s^{\triangleright} \in Q^{\triangleright} \text{ s.t., } i \in \mathrm{Activ}_{\mathrm{st}'}(b, s^{\triangleright}) \text{ and} \\ & (|\mathrm{Activ}_{\mathrm{st}'}(b, s^{\triangleright})| \leq 2) \text{ or } \big(|\mathrm{Activ}_{\mathrm{st}'}(b, s^{\triangleright})| > 2 \wedge i \in \{i_\ell^{b,s^{\triangleright}}, i_u^{b,s^{\triangleright}}\}\big) \\ \text{undefined} & \text{otherwise} \end{cases} \tag{5.10}$$

Then, observe that $\widetilde{\mathrm{st}'}$ is a $\triangleright$-safe state. From the discussion above, we obtain that $\mathcal{B}$ and $\mathcal{B}'$ still accept the same set of timed words. Hence we may conclude that in $\mathcal{B}'$, we can restrict to the *finite* index set $\mathrm{Ind}_{\mathit{fin}} = Act \times \{0, \ldots, n\}$ where $n = \max\{|Q^{\triangleleft}|, 2|Q^{\triangleright}|\}$. Consequently, $\mathcal{B}'$ uses finitely many states and clocks.

Theorem 5.12. *The timed automaton $\mathcal{B}'$ is finite. It has $B^{\mathcal{O}(|Proc|^2)}$ many clocks (for $B \geq 2$), and we have $\mathcal{L}_{tw}(\mathcal{B}') = \mathcal{L}_{tw}(\mathcal{B})$.*

Proof. We first note that the finiteness of the automaton follows immediately from the fact that Ind_{fin} is finite. By Lemma 5.6, we know that, for $B \geq 2$, $|Q^{\triangleleft}| = B^{\mathcal{O}(|Proc|^2)}$ and $|Q^{\triangleright}| = B^{\mathcal{O}(|Proc|^2)}$ and so we can conclude that the number of clocks is also bounded by $B^{\mathcal{O}(|Proc|^2)}$.

To prove that the automata accept the same timed language we will use the alternative definition of an accepting run of a timed automaton, which has moves with regard to time-elapse instead of time stamps.

We define a relation $\rightsquigarrow$ between configurations of $\mathcal{B}$ and $\mathcal{B}'$ as follows: $(\mathrm{st}_1, \nu_1) \rightsquigarrow (\mathrm{st}_2, \nu_2)$ if

(i) $\mathrm{st}_1 = (\overline{s}, \chi, \eta, \xi_1^{\triangleleft}, \xi_1^{\triangleright}, \gamma)$, $\mathrm{st}_2 = (\overline{s}, \chi, \eta, \xi_2^{\triangleleft}, \xi_2^{\triangleright}, \gamma)$

(ii) there exists $(b, i) \in \mathrm{dom}(\xi_1^{\triangleleft})$ such that $\nexists (b, j) \in \mathrm{dom}(\xi_1^{\triangleleft})$ with $\xi_1^{\triangleleft}(b, i) = \xi_1^{\triangleleft}(b, j)$ if and only if there exists $(b, k) \in \mathrm{dom}(\xi_2^{\triangleleft})$. And in this case $\xi_2^{\triangleleft}(b, k) = \xi_1^{\triangleleft}(b, i)$ and $\nu_1(z^{\triangleleft}_{(b,i)}) = \nu_2(z^{\triangleleft}_{(b,k)})$.

(iii) Similarly, there exists $(b, i) \in \mathrm{dom}(\xi_1^{\triangleright})$ such that $i \in \mathrm{Activ}_{\mathrm{st}}(b, s^{\triangleright})$ and either $(|\mathrm{Activ}_{\mathrm{st}'}(b, s^{\triangleright})| \leq 2)$ or $\big(|\mathrm{Activ}_{\mathrm{st}'}(b, s^{\triangleright})| > 2 \wedge i \in \{i_\ell, i_u\}\big)$ if and only if there exists $(b, k) \in \mathrm{dom}(\xi_2^{\triangleright})$. Again in this case $\xi_2^{\triangleright}(b, k) = \xi_1^{\triangleright}(b, i)$ and $\nu_1(z^{\triangleright}_{(b,i)}) = \nu_2(z^{\triangleright}_{(b,k)})$.

We show that the relation $\rightsquigarrow$ is a bisimulation. That is,

Claim 5.13. *Let* $(\mathrm{st}_1, \nu_1) \rightsquigarrow (\mathrm{st}_2, \nu_2)$. *Then,*

1. *for every move* $(\mathrm{st}_1, \nu_1) \xrightarrow{a,\tau} (\mathrm{st}_1', \nu_1')$ *there exists a move* $(\mathrm{st}_2, \nu_2) \xrightarrow{a,\tau} (\mathrm{st}_2', \nu_2')$ *such that* $(\mathrm{st}_1', \nu_1') \rightsquigarrow (\mathrm{st}_2', \nu_2')$ *and*

2. *conversely, for every move* $(\mathrm{st}_2, \nu_2) \xrightarrow{a,\tau} (\mathrm{st}_2', \nu_2')$ *there exists a move* $(\mathrm{st}_1, \nu_1) \xrightarrow{a,\tau} (\mathrm{st}_1', \nu_1')$ *such that* $(\mathrm{st}_1', \nu_1') \rightsquigarrow (\mathrm{st}_2', \nu_2')$

Proof. (1) Let $\mathrm{st}_1 = (\overline{s}, \chi, \eta, \xi_1^{\triangleleft}, \xi_1^{\triangleright}, \gamma)$ and $\mathrm{st}_2 = (\overline{s}, \chi, \eta, \xi_2^{\triangleleft}, \xi_2^{\triangleright}, \gamma)$. Then $(\mathrm{st}_1, \nu_1) \xrightarrow{a,\tau} (\mathrm{st}_1', \nu_1')$ with $\mathrm{st}_1' = (\overline{s}', \chi', \eta', \xi_1'^{\triangleleft}, \xi_1'^{\triangleright}, \gamma')$ if

$$(\mathrm{st}_1, \varphi, a, R, \mathrm{st}_1') \in \delta_{\mathcal{B}} \text{ for some } \varphi, R \tag{5.11}$$

$$\nu_1 + \tau \models \varphi \tag{5.12}$$

$$\nu_1' = (\nu_1 + \tau)[R \mapsto 0] \tag{5.13}$$

We can now define φ', R' by replacing each occurence of $z^{\triangleleft}_{(b,i)}$ (and $z^{\triangleright}_{(b,i)}$) in φ by $z^{\triangleleft}_{(b,k)}$ (respectively $z^{\triangleright}_{(b,k)}$) for some k given by condition (ii) (respectively, (iii)). Then, $(\mathrm{st}_2, \varphi', a, R', \mathrm{st}_2') \in \delta_{\mathcal{B}'}$ where $\mathrm{st}_2' = (\overline{s}', \chi', \eta', \xi_2'^{\triangleleft}, \xi_2'^{\triangleright}, \gamma')$ with $\xi_1'^{\triangleleft}$ and $\xi_2'^{\triangleleft}$ obtained from the definition of the respective modified transition relation.

Now, by Proposition 5.9 each prev-clock mentioned in φ has an image in φ' which by definition has the same constraint. Again, if φ mentions a next-clock then

either it itself is in φ' or there exists some other clocks in φ' whose upper guard and lower guards are stronger than it. We can conclude that $\nu_1 + \tau \models \varphi$ if and only if $\nu_2 + \tau \models \varphi'$ and $\nu_2' = (\nu_2 + \tau)[R' \mapsto 0]$. Thus, we have $(\mathrm{st}_2, \nu_2) \xrightarrow{a,\tau} (\mathrm{st}_2', \nu_2')$ and we are done once we see that $(\mathrm{st}_1', \nu_1') \rightsquigarrow (\mathrm{st}_2', \nu_2')$. Condition (i) is already true.

For Condition (ii), for $(b, i) \in \mathrm{dom}(\xi_1'^{\triangleleft})$ there are two cases to consider. First, it was defined here, i.e, $b = a$ and $i = \min(\mathbb{N} \setminus \mathrm{dom}(\xi_1^{\triangleleft}(a)))$. In this case, (b, i) is not $\triangleleft$-unsafe at st_1' if and only if (b, k) is not $\triangleleft$-unsafe at st_2', for $k = \min(\mathbb{N} \setminus \mathrm{dom}(\xi_2^{\triangleleft}(a)))$. Second, it was updated from the previous state (i.e, $(b, i) \in \mathrm{dom}(\xi_1^{\triangleleft})$). Now, if it was updated from the previous state st_1 and it is not $\triangleleft$-unsafe at st_1' then it was not $\triangleleft$-unsafe at st_1 either (because, otherwise there is $(b, j) \in \mathrm{dom}(\xi_1^{\triangleleft})$ such that $\xi_1^{\triangleleft}(b, i) = \xi_1^{\triangleleft}(b, j)$ which implies that $\delta^{\triangleleft}(\xi_1^{\triangleleft}(b, i), (a, \eta(a))) = \delta^{\triangleleft}(\xi_1^{\triangleleft}(b, j), (a, \eta(a)))$ which implies that (b, i) is $\triangleleft$-unsafe at st_1'). Then, as $\mathrm{st}_1 \rightsquigarrow \mathrm{st}_2$, there exists k such that $\xi_2^{\triangleleft}(b, k) = \xi_1^{\triangleleft}(b, i)$. Now, (b, k) cannot be $\triangleleft$-unsafe at st_2' since otherwise we obtain that (b, i) will be $\triangleleft$-unsafe at st_1'. Thus, $\xi_2'^{\triangleleft}(b, k) = \delta^{\triangleleft}(\xi_2^{\triangleleft}(b, k)(a, \eta(a))) = \delta^{\triangleleft}(\xi_1^{\triangleleft}(b, j)(a, \eta(a))) = \xi_1'^{\triangleleft}(b, i)$. Conversely, if there exists $(b, k) \in \mathrm{dom}(\xi_2'^{\triangleleft})$ and $\xi_2'^{\triangleleft}(b, k) = \xi_1'^{\triangleleft}(b, i)$, then there cannot exist $(b, j) \in \mathrm{dom}(\xi_1'^{\triangleleft})$ such that $\xi_1'^{\triangleleft}(b, j) = \xi_1'^{\triangleleft}(b, i)$. By similar arguments, we can conclude that Condition (iii) also holds.

(2) In the converse direction, for previous, next index (b, i) of st_1 which doesn't have a corresponding index in st_2, we use a fresh previous or next clock and use the same arguments as above. □

Finally, from these bisimulations, and the fact that the final states coincide, we may conclude that the timed languages are the same. □

Thus, from the above theorem and Theorem 5.7, we obtain the proof of the main result, i.e., Theorem 5.1, that we set out to prove in the beginning of this chapter.

6

Checking Conformance for Distributed Timed Specifications

In this chapter, we consider the specification model of a TCMSG that we introduced in Section 3.5. To begin with, we would like to reason about the set of behaviours of a TCMSG and examine conditions under which this set is regular. For this, we lift the notion of *locally synchronized* from MSGs to TCMSGs and show that this is a sufficient condition to ensure regularity. We now consider some other problems that arise naturally in this context. We state them in a general setting though we will be able solve them only with some restrictions, for instance, locally synchronized.

6.1 The problem statements

We are interested in comparing these behaviours to those generated by an implementation model of a TMPA. Thus, given a TCMSG $\mathfrak{G}$ and a TMPA $\mathcal{A}$, we address the question of checking whether the implementation $\mathcal{A}$ *conforms* to the specification $\mathfrak{G}$. This breaks up as two natural problems.

Consistency For each TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$, is there a TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$ such that T realizes $\mathfrak{M}$?

Coverage For each TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, is there a TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$?

Let us examine these problems in some detail before going further.

The Consistency Problem

Consistency asks if all the behaviours of a TMPA $\mathcal{A}$ are legal with respect to a TCMSG specification $\mathfrak{G}$.

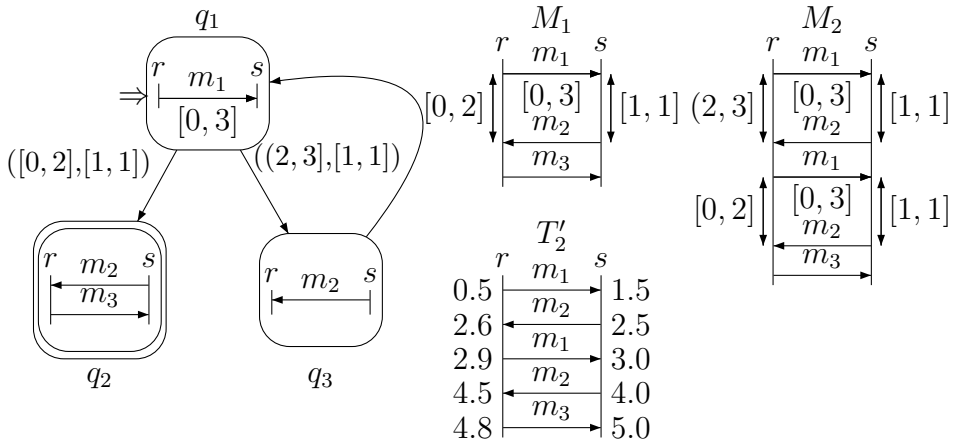


Figure 6.1: A TCMSG and some TCMSCs that it generates

Example 30. For instance, consider the TCMSG $\mathfrak{G}$ in Figure 6.1 and the TMPA $\mathcal{A}$ in Figure 6.2. Then it is not difficult to check that for any TMSC T generated by $\mathcal{A}$ there is a TCMSC $\mathfrak{M}$ generated by $\mathfrak{G}$ such that T realizes $\mathfrak{M}$.

Thus, consistency corresponds to asking if the timed language of MSC linearizations associated with the TMPA is included in the timed language of the MSC linearizations associated with the TCMSG. In general, this problem is undecidable, even for regular timed specifications [5], i.e, bounded TMPAs. However, if we additionally restrict ourselves to locally sychronized TCMSGs, then we can solve the timed language inclusion problem in this context. This is done by exploiting the fact that the timing constraints in locally synchronized TCMSGs correspond to using clocks in a very restricted way.

The Coverage Problem

Let $\mathfrak{G}$ be a TCMSG and $\mathcal{A}$ a TMPA. The *coverage problem* for $\mathfrak{G}$ and $\mathcal{A}$ is to determine whether for each TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, there is a TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$. This is a natural verification problem when we interpret TCMSGs as incomplete positive specifications.

Example 31. Again, we consider the TCMSG $\mathfrak{G}$ in Figure 6.1 and the TMPA from Figure 6.2. Then, $\mathfrak{M}_1$ and $\mathfrak{M}_2$ (Figure 6.1) are TCMSCs in $\mathcal{L}_{TC}(\mathfrak{G})$, generated by paths q_1q_2 and $q_1q_3q_1q_2$, respectively. We observe that these are realized by the TMSCs T_1 and T_2 (Figure 6.2) in $\mathcal{L}_{time}(\mathcal{A})$.

Coverage in the untimed case In the untimed case, the corresponding problem of scenario matching considered in [36,52], asks whether $\mathcal{L}_{MSC}(\mathcal{G}) \subseteq \mathcal{L}_{MSC}(\mathcal{A})$ where $\mathcal{G}$ is an MSG and $\mathcal{A}$ is an MPA. In the timed case, we cannot reduce coverage to language inclusion of timed MSCs. A TCMSC $\mathfrak{M}$ represents an infinite

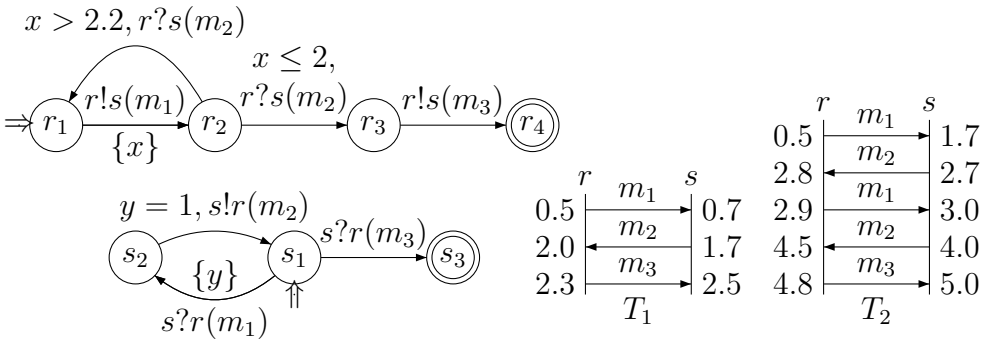


Figure 6.2: A timed MPA and some timed MSCs that it recognizes

family of TMSCs, each of which realizes $\mathfrak{M}$. However, the implementation need not, in general, permit all these realizations. For instance, the TMPA in Figure 6.2 will not exhibit any TMSC realizing $\mathfrak{M}_2$ from Figure 6.1 where the time difference between the first two p-events is 2.1, such as in the TMSC T_2' in Figure 6.1.

Problems with a game-theoretic approach to coverage Another plausible approach is to treat this as a timed game between Spoiler, who picks a path in the TCMSG $\mathfrak{G}$, and Duplicator, who picks a TMSC in $\mathcal{L}_{time}(\mathcal{A})$ that realizes the TCMSC generated along the path chosen by Spoiler. At each step, Spoiler adds a node to the path in $\mathfrak{G}$. Duplicator has to match this move by extending the current TMSC so that it stays in $\mathcal{L}_{time}(\mathcal{A})$ and realizes the TCMSC described by the extended path. However, a winning strategy in this game would have the following property: if two paths π_1 and π_2 have a common prefix π, then the TMSC generated by Duplicator for the prefix π must be the same for the plays in which Spoiler generates π_1 and π_2. Notice that the paths that generate $\mathfrak{M}_1$ and $\mathfrak{M}_2$ in Figure 6.1 share a common prefix, namely q_1. In any TMSC that realizes $\mathfrak{M}_1$, message m_1 must be delivered within 1 time unit whereas in any TMSC that realizes $\mathfrak{M}_2$, m_1 can only reach after 1 time unit. Hence, any TMSC that realizes $\Phi(q_1)$ and can be extended to cover $\mathfrak{M}_1$ cannot simultaneously be extended to cover $\mathfrak{M}_2$. In other words, the game-theoretic formulation introduces too strict a correlation between the TMSCs covering different paths through the TCMSG.

These observations suggest that traditional approaches for scenario matching in the untimed case do not generalize to the coverage problem in the timed case. In the last section of this chapter, we formulate a solution to the coverage problem for a restricted class of locally synchronized TCMSGs.

6.2 An extended event clock automaton – the MSC-ECA

To understand the operational behaviour of a TCMSG it is useful to look at it as a global system and examine the set of timed linearizations it accepts. It is natural to consider the global system as an infinite-state timed transition system. In this section, we introduce the MSC-event clock automaton which we develop as an (extended) event clock automaton designed towards recognizing timed linearizations of MSCs.

In the next section, we will provide the translation from TCMSGs to MSC-ECAs thus providing a global semantics (in terms of timed linearizations) for the TCMSG. This translation will allow us to prove the results in the following sections in an elegant manner.

We now start by defining the extension of event clock automata, namely 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 by $\mathcal{C} = (Q, Act, \delta, q_0, F)$, where Q is a set of states, with $q_0 \in Q$ the initial state and $F \subseteq Q$ the set of final states. The alphabet is the set of actions Act. A transition in δ is of the form (q, φ, a, q') where $q, q' \in Q, a \in Act$ and φ is a conjunction of event clock guards, which are of two types: either $\mathrm{Y}_p^k \in I$ or $\mathrm{Msg}^{-1} \in I$. An MSC-ECA is said to be *finite* if it has finitely many states.

We interpret these guards over timed words. Let $\sigma = (a_1, t_1) \ldots (a_n, t_n) \in \mathrm{TW}_{Act}$. Then at a position $1 \leq j \leq n$, we define

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

(D2) $\sigma, j \models \mathrm{Msg}^{-1} \in I$ if 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 $a_k \in p!q$ to mean $a_k = p!q(m')$ for some $m' \in \mathcal{M}$). In other words, a_j is a receive action and the time elapsed since the occurence of its matching send action a_i is in the interval I.

In both the above definitions, note that the action a_i is uniquely defined, i.e., there is atmost one position i that matches a given position j.

Now, we define runs of $\mathcal{C}$ over timed words. For a timed word $\sigma = (a_1, t_1) \ldots (a_n, t_n)$, 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 $q = q_0 \xrightarrow{\varphi_1, a_1} \ldots \xrightarrow{\varphi_n, a_n} q_n$ such that for all j, $1 \leq j \leq n$, $\sigma, j \models \varphi_j$. The timed word σ is said to be accepted if it has a run from the initial to some

final state in F. We denote by $\mathcal{L}_{tw}(\mathcal{C})$ the set of timed words accepted by the MSC-ECA $\mathcal{C}$.

In what follows, we will call a timed word σ *well-formed* if every receive action has a corresponding send action on its left.

6.2.1 Translation from MSC-ECA to TA

Let $\mathcal{C} = (Q, Act, \delta, q_0, F)$ be an MSC-ECA. Let $B \in \mathbb{N}_{>0}$ be a positive integer which will act as a bound in what follows. Also let $K = max\{k \mid \mathrm{Y}_p^k \in I$ occurs in some guard in $\delta\}$. We define a finite timed automaton $\mathcal{B}_{\mathcal{C}}^B$ from $\mathcal{C}$ as follows:

A state of $\mathcal{B}_{\mathcal{C}}^B$ is either a dead state denoted $\perp$ or a tuple of the form $\mathfrak{s} = (s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$ where,

- $s \in Q$
- $\overline{b} = (b_p)_{p \in Proc} \in \{0,1\}^{Proc}$: We will have $b_p = 1$ 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.
- $\overline{\alpha} = (\alpha_{p,q})_{p,q \in Proc}$ where $\alpha_{p,q} \in \{0, \ldots, B\}$: $\alpha_{p,q}$ is the number of events in $q?p$ modulo $B+1$.
- $\overline{\beta} = (\beta_{p,q})_{p,q \in Proc}$ where $\beta_{p,q} \in \{0, \ldots B\}$: $\beta_{p,q}$ is the number of events in $p!q$ modulo $B+1$.

We say that the Channel (p,q) is *empty* if $\alpha_{p,q} = \beta_{p,q}$ and *full* if $\beta_{p,q} = (\alpha_{p,q} + B) \mod (B+1)$. 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,

$$\begin{aligned} Y &= \{y_p^i \mid p \in Proc, 0 \leq i < K\} \\ Z &= \{z_{p,q}^i \mid p, q \in Proc, 0 \leq i \leq B\} \end{aligned}$$

The idea is that we will reset the clock y_p^i when executing the i^{th} p-event modulo K. Also, $z_{p,q}^i$ will be reset when executing the i^{th} $p!q$ event modulo $B+1$.

The set of transitions $\delta_{\mathcal{B}_{\mathcal{C}}^B}$ is defined as follows: Assume $s \xrightarrow{\varphi, a} s'$ in $\mathcal{C}$ with $a \in Act_p$. Then, we have three types of transitions in $\mathcal{B}_{\mathcal{C}}^B$:

(Tr1) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\text{true}, a, \emptyset} \perp$ is in $\mathcal{B}_{\mathcal{C}}^B$ if

- either $a \in p!q$ and Channel (p,q) is full, i.e., the bound was exceeded.

- or $a \in p?q$ and Channel (p,q) is empty i.e., a receive occurs with no matching send in the queue.

(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. $n'_p = (n_p + 1) \mod K$ and $n'_r = n_r$ for $r \neq p$.
3. if $a \in p!q$, then
   (i) $\beta'_{p,q} = (\beta_{p,q} + 1) \mod (B+1)$ and $\beta'_{p',q'} = \beta_{p',q'}$ for $(p',q') \neq (p,q)$. Also $\overline{\alpha}' = \overline{\alpha}$.
   (ii) $R = \{y_p^{n'_p}, z_{p,q}^{\beta'_{p,q}}\}$
   (iii) φ' is φ where $\mathrm{Y}_p^k \in I$ is replaced with

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

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

(Tr3) $\perp \xrightarrow{\text{true}, a, \emptyset} \perp$ is in $\mathcal{B}^B_{\mathcal{C}}$ for all $a \in Act$.

We can immediately observe some invariant properties that are maintained by the above transitions.

Remark 6.1. Let $\mathfrak{s}_0 \xrightarrow{\varphi_1, a_1, R_1} \ldots \xrightarrow{\varphi_m, a_m, R_m} \mathfrak{s}_m$ for $m \geq 0$ be a path in $\mathcal{B}^B_{\mathcal{C}}$ from the initial state $\mathfrak{s}_0$ to some state $\mathfrak{s}_m \neq \perp$. 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. $n_p = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in Act_p\}| \mod K$
3. $\alpha_{p,q} = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in q?p\}| \mod (B+1)$
4. $\beta_{p,q} = |\{\ell \mid 1 \leq \ell \leq m \wedge a_\ell \in p!q\}| \mod (B+1)$

Remark 6.2. On the other hand, suppose $\mathfrak{s}_0 \xrightarrow{\varphi_1,a_1,R_1} \dots \xrightarrow{\varphi_m,a_m,R_m} \mathfrak{s}_m$ 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 well-formed or it exceeds the bound B for some channel. Indeed, we can reach $q_i = \bot$ from $q_{i-1} \neq \bot$ only by applying transition *(Tr1)*, where either boundedness or well-formedness is violated. Further, if we reach $\bot$, then only transition *(Tr3)* applies and hence we remain in $\bot$.

Now, using the above properties in Remark 6.1, we can prove the following lemmas relating runs of $\mathcal{C}$ and $\mathcal{B}_{\mathcal{C}}^B$. First, for a run of $\mathcal{C}$ on σ passing through the states $s_1 \dots s_n$, we call a run of $\mathcal{B}_{\mathcal{C}}^B$ its *corresponding run* on σ, if it passes through states $\mathfrak{s}_i = (s_i, \overline{b}_i, \overline{n}_i, \overline{\alpha}_i, \overline{\beta}_i)$ for $i \in \{0, \dots, n\}$, for some values of $\overline{b}_i, \overline{n}_i, \overline{\alpha}_i, \overline{\beta}_i$ and guards are obtained by using *(Tr2)* 3.(iii) and 4.(iii).

Lemma 6.3. *For every run of $\mathcal{C}$ on a timed word σ, there exists a unique corresponding run of $\mathcal{B}_{\mathcal{C}}^B$ on σ.*

Proof. Consider a run of $\mathcal{C}$ on $\sigma = (a_1, t_1) \dots (a_m, t_m)$. Then there exists $\pi = s_0 \xrightarrow{\varphi_1,a_1} s_1 \xrightarrow{\varphi_2,a_2} \dots \xrightarrow{\varphi_m,a_m} s_m$ in $\mathcal{C}$ such that $\sigma, i \models \varphi_i$ for all $i \in \{1, \dots, m\}$. Then, we inductively construct the unique corresponding run in $\mathcal{B}_{\mathcal{C}}^B$ over the same timed word which starts with the initial state $(\mathfrak{s}_0, \nu_0)$ where $\mathfrak{s}_0$ is the initial state of $\mathcal{B}_{\mathcal{C}}^B$ and ν_0 is the valuation mapping all clocks to 0. Suppose we have constructed the path till $(\mathfrak{s}_{i-1}, \nu_{i-1})$ with $\mathfrak{s}_{i-1} = \bot$ or $\mathfrak{s}_{i-1} = (s_{i-1}, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$, then for the next transition we consider the following cases.

Suppose $\mathfrak{s}_{i-1} = \bot$, then there exists a unique transition *(Tr3)* and so we have $\mathfrak{s}_i = \bot$. Otherwise $\mathfrak{s}_{i-1} = (s_{i-1}, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$. Then, if $a_i \in p!q$ and the channel (p, q) is full or $a_i = p?q$ and the channel is empty, we have the unique transition *(Tr1)* and again $\mathfrak{s}_i = \bot$. Otherwise, we can observe that using $s_{i-1} \xrightarrow{\varphi_i,a_i} s_i$ in $\mathcal{C}$, we obtain the unique transition $\mathfrak{s}_{i-1} = (s_{i-1}, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\varphi_i',a_i,R_i} (s_i, \overline{b}', \overline{n}', \overline{\alpha}', \overline{\beta}') = \mathfrak{s}_i$ as defined by *(Tr2)*. This constructs the (unique) corresponding path in $\mathcal{B}_{\mathcal{C}}^B$. Letting, $\nu_i = (\nu_{i-1} + t_i - t_{i-1})[R_i \to 0]$ this defines a run of $\mathcal{B}_{\mathcal{C}}^B$ on σ: $(\mathfrak{s}_0, \nu_0) \xrightarrow{a_1,t_1} \dots \xrightarrow{a_m,t_m} (\mathfrak{s}_m, \nu_m)$, if we show that $\nu_{i-1} + t_i - t_{i-1} \models \varphi_i'$ for all i. There are three cases:

1. φ_i' contains $z_{p,q}^k \in I$ where $k = \alpha_{p,q}' = |\{a_\ell \mid 1 \leq \ell \leq i \wedge a_\ell \in q?p\}| \mod (B+1)$. By 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, $k = |\{a_\ell \mid 1 \leq \ell \leq j \wedge a_\ell \in p!q\}| \mod (B+1)$. 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 get $\mathrm{Msg}^{-1} \in I$ is replaced with $z_{p,q}^k \in I$ in φ_i'. Moreover, $z_{p,q}^k \notin R_\ell$ for $j < \ell \leq i$. This follows since 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, $z^k_{p,q} \in R_j$ implies $\nu_j(z^k_{p,q}) = 0$ and $z^k_{p,q} \notin R_\ell$ for $j < \ell \leq i$ implies that $(\nu_{i-1} + t_i - t_{i-1})(z^k_{p,q}) = \nu_j(z^k_{p,q}) + t_i - t_j = t_i - t_j$. So, we have $(\nu_{i-1} + t_i - t_{i-1}) \models (z^k_{p,q} \in I)$.

2. We will show that φ' cannot contain false. If φ' contains false, then $b_p = 0$ and there exists $\mathrm{Y}^k_p \in I$ 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. Which is a contradiction with $\sigma, i \models \mathrm{Y}^k_p \in I$ and so this cannot happen.

3. φ'_i contains $y^\ell_p \in I$, Then $\ell = (K + n'_p - k) \mod K$ and $\mathrm{Y}^k_p \in I$ 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^ℓ_p 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 $(\nu_{i-1} + t_i - t_{i-1})(y^\ell_p) = \nu_j(y^\ell_p) + t_i - t_j = t_i - t_j \in I$. Thus, $(\nu_{i-1} + t_i - t_{i-1}) \models y^\ell_p \in I$.

□

The next property is the converse of the above but in this case, we see that we have to restrict the runs as follows:

Lemma 6.4. *If $\mathcal{B}^B_\mathcal{C}$ has a run on σ which does not end in the dead state $\perp$, then $\mathcal{C}$ has a run on σ as well.*

Proof. In *(Tr2)*, given $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$, a and φ we define φ' and hence the transition of $\mathcal{B}^B_\mathcal{C}$. But observe that if we know $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$, a and the resultant φ', then we can uniquely recover φ as well as the transition on $\mathcal{C}$. This allows us to construct the path in $\mathcal{C}$ using the given path in $\mathcal{B}^B_\mathcal{C}$ on the word σ. Indeed this works only since we assume that the path in $\mathcal{B}^B_\mathcal{C}$ uses only transitions *(Tr2)*. Thus, given the path in $\mathcal{B}^B_\mathcal{C}$

$$\mathfrak{s}_0 \xrightarrow{\varphi'_1, a_1, R_1} \mathfrak{s}_2 \ldots \xrightarrow{\varphi'_n, a_n, R_n} \mathfrak{s}_n$$

we obtain the path in $\mathcal{C}$,

$$s_0 \xrightarrow{\varphi_1, a_1} s_1 \ldots \xrightarrow{\varphi_n, a_n} s_n$$

We need to show that if the former path defines a timed run so does the latter, i.e., $\sigma, i \models \varphi_i$ for all i. Again 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 all the cases. □

We can define different notions of acceptance (i.e., final state) on $\mathcal{B}^B_\mathcal{C}$ constructed from $\mathcal{C}$ to derive different results.

Corollary 6.5. *Given* $\mathcal{C} = (Q, Act, \delta, q_0, F)$*, if we construct* $\mathcal{B}^B_{\mathcal{C}} = (Q', Act, (Y \cup Z), \delta_{\mathcal{B}^B_{\mathcal{C}}}, F')$ *as described above and*

1. *if* $F' = \{(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \mid s \in F\}$*, then* $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}}) = \mathcal{L}_{tw}(\mathcal{C}) \cap \{\sigma \in \mathrm{TW}_{Act} \mid \sigma$ *is* B*-bounded and well-formed*$\}$*.*

2. *if* $\mathcal{C}$ *is complete, i.e., it has a run on every timed word over* Act*, and* $F' = \{\bot\}$*, then* $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}}) = \{\sigma \in \mathrm{TW}_{Act} \mid$ *either* σ *is not* B*-bounded or* σ *is not well-formed*$\}$*.*

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

Proof. Statement 1 follows from the above lemmas and Remark 6.2. The statement 2 follows directly from Remark 6.2. □

6.2.2 A universal automaton

To justify that the MSC-ECA that we introduced have nice properties, we will now prove that they can be determinized. We obtain this by constructing a deterministic and complete version of any given MSC-ECA. 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. Thus, a state is of the form $X \subseteq Q$. Before defining the transitions, we introduce some notations. First, if $t \in \delta$ is a transition of $\mathcal{C}$, then we let q_t, q'_t respectively denote its source and target states, φ_t denotes its guard and a_t the action. Now, for a state X and an action a, we define $T(X, a) = \{t \in \delta \mid q_t \in X, a_t = a\}$. Then, for some $T' \subseteq T(X, a) = T$, we have the macros:

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

$$\mathrm{target}(T') = \{q' \in Q \mid q' = q'_t \text{ for some } t \in T'\}$$

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

- $\varphi = \varphi(T', T)$
- $X' = \mathrm{target}(T')$

Indeed, we note that once we have fixed X, a and the set T', the transition is uniquely defined. Also note that for $X = \emptyset$, we have $T(X, a) = \emptyset$ and the only transition possible is $\emptyset \xrightarrow{\text{true}, a} \emptyset$. A run is again defined as before on a timed word $\sigma = (a_1, t_1) \dots (a_n, t_n)$ as a sequence of transitions $X_0 \xrightarrow{\varphi_1, a_1} X_1 \dots \xrightarrow{\varphi_n, a_n} X_n$ such that $\sigma, j \models \varphi_j$ for all j. The crucial property of this automaton is that it is deterministic and complete as shown by the following lemma.

Lemma 6.6. *For any timed word $\sigma = (a_1, t_1) \dots (a_n, t_n)$, there exists a unique run of $\mathcal{C}^{univ}$ on σ starting from $X_0 = \{q_0\}$.*

Proof. Given $\sigma = (a_1, t_1) \dots (a_n, t_n)$ and $X_0 = \{q_0\}$, for $j \in \{1, \dots, n\}$, we define inductively $T_j = T(X_{j-1}, a_j)$, $T'_j = \{t \in T_j \mid \sigma, j \models \varphi_t\}$ and $X_j = \text{target}(T'_j)$. 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

$$\sigma, j \models \varphi(T', T_j) \text{ if and only if } T' = T'_j \tag{6.1}$$

Using the "if" part above, we immediately obtain $X_0 \xrightarrow{\varphi(T'_1, T_1), a_1} X_1 \dots \xrightarrow{\varphi(T'_n, T_n), a_n} X_n$ 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 $X_0 \xrightarrow{\varphi_1, a_1} X'_1 \xrightarrow{\varphi_2, a_2} \dots X'_n$ 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 $X'_j = \text{target}(T')$. Thus, $\sigma, j \models \varphi(T', T'_j)$ which by Equation 6.1 implies that $T' = T'_j$. Thus, we conclude $\varphi_j = \varphi(T'_j, T_j)$ and $X'_j = \text{target}(T') = \text{target}(T'_j) = X_j$. □

Now we can show

Lemma 6.7. *If $X_0 \xrightarrow{\varphi_1, a_1} \dots \xrightarrow{\varphi_n, a_n} X_n$ is the run of $\mathcal{C}^{univ}$ on $\sigma = (a_1, t_1) \dots (a_n, t_n)$, then $X_n = \{q \in Q \mid q_0 \xrightarrow{\sigma} q \text{ in } \mathcal{C}\}$*

Proof. (1) In one direction, if $q_0 \xrightarrow{\sigma} q$ in $\mathcal{C}$, let $q_0 \xrightarrow{\varphi'_1, a_1} \dots \xrightarrow{\varphi'_n, a_n} q_n = q$ with $\sigma, j \models \varphi'_j$ for $1 \leq j \leq n$. Using the proof of the above lemma, we will show that for all $j \in \{0, \dots, n\}$, $q_j \in X_j$. 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 $q_j \in X_j = \text{target}(T'_j)$.

(2) In the other direction, for all j and for all $q_j \in X_j$ we want to show that $q_0 \xrightarrow{a_1, t_1} \dots \xrightarrow{a_j, t_j} q_j$ 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 Induction hypothesis we have $q_0 \xrightarrow{a_1, t_1} \dots \xrightarrow{a_{j-1}, t_{j-1}} q_{j-1}$. This implies that $q_0 \xrightarrow{a_1, t_1} \dots \xrightarrow{a_{j-1}, t_{j-1}} q_{j-1} \xrightarrow{a_j, t_j} q_j$ is a run of $\mathcal{C}$. □

Note that if $\mathcal{C}$ is finite so is $\mathcal{C}^{univ}$. Indeed the description of $\mathcal{C}^{univ}$ is not complete since we have not specified the final states. Given $\mathcal{C} = (Q, Act, \delta, q_0, F)$, we will now construct two universal (i.e., deterministic and complete) MSC-ECA as detailed above $\mathcal{C}_i^{univ} = (2^Q, Act, \Delta, \{q_0\}, F_i)$ for $i = \{1, 2\}$ differing only in the set of final states, defined as: $F_1 = \{X \in 2^Q \mid \exists s \in (F \cap X)\}$ and $F_2 = 2^Q \setminus F_1$. Then, the following corollary directly follows from the above lemmas,

Corollary 6.8. *We have $\mathcal{L}_{tw}(\mathcal{C}_1^{univ}) = \mathcal{L}_{tw}(\mathcal{C})$ and $\mathcal{L}_{tw}(\mathcal{C}_2^{univ}) = \{\sigma \in \text{TW}_{Act} \mid \sigma \notin \mathcal{L}_{tw}(\mathcal{C})\}$.*

6.3 Global semantics for TCMSGs over timed linearizations

We begin by defining *the* canonical global semantics of TCMSG in terms of timed linearizations, i.e.,

Definition 6.9. *The* timed word language *of a TCMSG* $\mathfrak{G}$ *is defined to be set of timed linearizations it accepts, i.e.,* $\mathcal{L}_{tw}(\mathfrak{G}) = \{\sigma \in \mathrm{TW}_{Act} \mid \sigma$ *is a timed linearization of some TMSC* T *over Act, such that* T *realizes some* $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})\}$.

However, this definition does not help us understand or reason about properties of timed languages accepted by the TCMSG. For this, we would prefer to have an equivalent automaton which generates the same language. In this section, from a TCMSG, we construct an MSC-ECA (with infinite states) which accepts exactly the same set of timed linearizations.

We start with a definition and a remark. For an MSC $M = (E, \leq, \lambda)$ over Act, we define 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 from Section 2.2.6 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 $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$, where $G = (V, \to, v_{in}, V_F)$, 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,

$$\varphi = \left(\bigwedge_{e' \in E^u, I \in \mathcal{I} | \tau^u(e', e) = I} \varphi(u, e', e, I) \right) \wedge \varphi^{edge} \text{ where,} \tag{6.2}$$

$$\varphi(u, e', e, I) = \begin{cases} \text{Msg}^{-1} \in I & \text{if } \exists p, q, p \neq q \text{ s.t. } e' <^u_{qp} e \\ \text{Y}^k_p \in I & \text{if } e, e' \in E^u_p, |\{e'' \in E^u_p \mid e' \leq^u_{pp} e'' <^u_{pp} e\}| = k \end{cases} \tag{6.3}$$

$$\text{and } \varphi^{edge} = \begin{cases} \text{Y}^1_p \in I & \text{if } \rho = \rho' u', \text{ and for some } p \in \mathit{Proc}, \text{ we have} \\ & \mathit{EdgeC}((u', u), p) = I \text{ and } e = \mathit{min}(E^u_p) \\ \text{true} & \text{otherwise} \end{cases} \tag{6.4}$$

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 $\sigma = (a_1, t_1) \ldots (a_n, t_n)$ is a sequence of transitions $\overline{s}_0 \xrightarrow{\varphi_1, a_1} \ldots \xrightarrow{\varphi_n, a_n} \overline{s}_n$ 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 6.10. *We have the following relation between the timed languages of* $\mathcal{C}_{\mathfrak{G}}$ *and* $\mathfrak{G}$*:* $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathfrak{G}) = \{\sigma \mid \sigma \text{ is a timed linearization of some TMSC } T \text{ over } \mathit{Act}, \text{ such that } T \text{ realizes some } \mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})\}$.

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

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

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, $\pi_0 = \ldots = \pi_n = \pi$ (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 $i \in \{1, \ldots, n\}$, $C_i = C_{i-1} \cup \{(e_i, \rho_i)\}$ is a cut of M^π. Moreover, $C_n = E^\pi$ since $\overline{s}_n$ is final. From this we get that $(e_1, \rho_1) \ldots (e_n, \rho_n)$ is a linearization of M^π and $\lambda^\pi(e_i, \rho_i) = a_i$ for all $i \in \{1, \ldots, n\}$. Now consider the TMSC $T = (M^\pi, t)$ where we define t by $t((e_i, \rho_i)) = t_i$. Thus, $(a_1, t_1) \ldots (a_n, t_n)$ is a timed linearization 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 realizes $\mathfrak{M}^\pi$. That is, for all $((e_i, \rho_i), (e_j, \rho_j)) \in \text{dom}(\tau^\pi)$, 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 $\varphi_j^{edge} = (\mathrm{Y}_p^1 \in I)$. 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 $\pi = v_1 \dots v_m$ 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 $\sigma = (a_1, t_1) \dots (a_n, t_n)$ is a timed linearization of $T = (M^\pi, t)$ and T realizes $\mathfrak{M}$. Then first we observe that $a_1 \dots a_n \in \mathrm{lin}(M^\pi)$ and so there is $(e_1, \rho_1) \dots (e_n, \rho_n)$ a linearization 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 $C_i = \{(e_1, \rho_1) \dots (e_i, \rho_i)\}$ 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., $\varphi^{edge} = (\mathrm{Y}_p^1 \in I)$ for some $p \in \mathit{Proc}$. In this case, by Condition 6.4, $\rho_i = \rho' v' v$, 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 realizes $\mathfrak{M}$, 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 $\sigma, i \models \varphi^{edge}$.
- 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 realizes $\mathfrak{M}$, $t(e_i, \rho u) - t(e_j, \rho u) = t_i - t_j \in I$. By Condition 6.3, if the constraint is of

the form $\text{Msg}^{-1} \in I$, then $e_j <^u_{qp} e_i$ and otherwise the constraint is of the form $\text{Y}^k_p \in I$ 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 $\overline{s}_0 \xrightarrow{\varphi_1, a_1} \ldots \xrightarrow{\varphi_n, a_n} \overline{s}_n$ is a run of the global system on σ. Finally, the run ends in a global final state, since σ is a full linearization of M^π. Thus, our proof is complete. □

6.4 Regularity for locally synchronized TCMSGs

The first question we ask is if the set of timed linearizations of a time-constrained message sequence graph is regular. We claim that the locally synchronized condition that we defined for MSGs in Section 2.2.6 is already good enough to ensure this. We start by restating Definition 2.18 in the TCMSG case,

Definition 6.11. *A TCMSG $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ is said to be* locally synchronized *if for every (simple) loop π in G, the TCMSC generated by π, $\mathfrak{M}_\pi = (M_\pi, \tau_\pi)$ is such that the MSC M_π is com-connected.*

Thus, we can state the main result of this section as follows.

Theorem 6.12. *If $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ is locally synchronized TCMSG, then there exists a finite MSC-ECA $\mathcal{C}$, such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$.*

Corollary 6.13. *If $\mathfrak{G}$ is a locally synchronized TCMSG, then there exists a finite timed automaton which accepts the timed language $\mathcal{L}_{tw}(\mathfrak{G})$.*

Proof. By the above theorem, from $\mathfrak{G}$, we have a finite MSC-ECA $\mathcal{C}$ such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G}) = \{\sigma \mid \sigma$ is a timed linearization of some TMSC T over Act, such that T realizes some $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})\}$. Thus, every $\sigma \in \mathcal{L}_{tw}(\mathcal{C})$ is well-formed. Further, since $\mathfrak{G}$ is locally synchronized, there exists bound $B \in \mathbb{N}_{>0}$ such that each timed linearization σ is B-bounded. Thus, every $\sigma \in \mathcal{L}_{tw}(\mathcal{C})$ is B-bounded. Now by Corollary 6.5(1), we obtain a timed automaton $\mathcal{B}^B_{\mathcal{C}}$ such that $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}}) = \mathcal{L}_{tw}(\mathcal{C})$. In fact, every $\sigma \in \mathcal{L}_{tw}(\mathcal{C})$ is complete as well and so we could have instead used Corollary 6.5(2) to conclude the above. The timed automaton $\mathcal{B}^B_{\mathcal{C}}$ is finite since the MSC-ECA $\mathcal{C}$ is finite which completes our proof. □

The above result in the untimed case has been stated and proved in different ways [8, 23, 27, 51] which are interesting in their own respect. We describe below a fully self-contained proof of this result in the timed version. By keeping it self-contained, we are able to modify the construction easily to show the results in the following sections.

6.4.1 The gap semantics

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 last part of this 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.

6.4.2 Removing unexecuted nodes

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 $(u_1 \dots u_n, C)$ into $(u_1, c_1) \dots (u_n, c_n)$. Formally, we define the map Φ which we call *stratification* as follows:

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

where each $c_i \subseteq E^{u_i}$ is defined by $c_i = \{e \in E^{u_i} \mid (e, u_1 \dots u_i) \in C\}$. 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 $C = \{(e, u_1 \dots u_i) \in E^{u_1 \dots u_n} \mid e \in c_i\}$.

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 $e \in E^u_p$ then for all $\alpha'_1(u',c') \preceq \alpha_1$, we have $E^{u'}_p \subseteq c'$.

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 $u \in V$, $OProc(u) = \{p \in Proc \mid E^u_p \neq \emptyset\}$ denotes the set of processes that participate (*occur*) in u. This is extended to V^* as a morphism. Also, with $OProc(u,c) = OProc(u)$ and $OProc(\#) = \emptyset$ it extends to Γ^*. In addition, for $\beta \in \Gamma^*$, $EProc(\beta)$ denoting the set of *executed* events in β, is given by the morphism defined by $EProc((u,c)) = \{p \in Proc \mid E^u_p \cap c \neq \emptyset\}$, $EProc(\#) = \emptyset$.

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, $OProc(u) \cap EProc(\alpha_2) = \emptyset$.

(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,

$$\varphi = \left(\bigwedge_{e' \in E^u, I \in \mathcal{I} | \tau^u(e',e) = I} \varphi(u, e', e, I) \right) \wedge \varphi^{edge} \text{ where,} \tag{6.5}$$

$$\varphi(u,e',e,I) = \begin{cases} \mathrm{Msg}^{-1} \in I & \text{if } \exists p,q, p \neq q \text{ s.t. } e' <^u_{qp} e \\ \mathrm{Y}^k_p \in I & \text{if } e, e' \in E^u_p, |\{e'' \in E^u_p \mid e' \leq^u_{pp} e'' <^u_{pp} e\}| = k \end{cases} \tag{6.6}$$

$$\text{and } \varphi^{edge} = \begin{cases} \mathrm{Y}^1_p \in I & \text{if } \beta_1 = \beta_1'(u', c'') \text{ and for some } p \in \mathit{Proc}, \text{ we have} \\ & \mathit{EdgeC}((u',u),p) = I \text{ and } e = \mathit{min}(E^u_p) \\ \text{true} & \text{otherwise} \end{cases} \tag{6.7}$$

(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 $u \to u'$ 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 6.14. *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 6.15. $\mathcal{L}_{tw}(\mathcal{C}^{\#}_{\mathfrak{G}}) = \mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}})$

Proof. We consider a morphism $\Psi : \mathit{ExtNodes}^* \to \Gamma^*$ 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,

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

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

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

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

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

- Either $c_j \neq \emptyset$. Then we observe that $\beta_{i-1} = \Upsilon(\overline{s}_{i-1}) = \alpha_1(u_j, c_j)\alpha_2$ where $\alpha_1 = (\triangleright, \emptyset)\mathit{Red}_{\#}(\Psi((u_1, c_1) \dots (u_{j-1}, c_{j-1})))$ and $\alpha_2 = \mathit{Red}_{\#}(\Psi((u_{j+1}, c_{j+1}) \dots (u_m, c_m)))(\triangleleft, \emptyset)$. Then, we observe that $(e, u_1 \dots u_j)$ 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 $c'_j = c_j \cup \{e\}$. From Conditions (6.2–6.4) and (6.5–6.7), we deduce that $\varphi' = \varphi$. Then, $\Phi(\overline{s}_i) = (u_1, c_1) \dots (u_j, c'_j) \dots (u_m, c_m)$ 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 $\alpha_1 \# = (\triangleright, \emptyset) Red_{\#}(\Psi((u_1, c_1) \dots (u_j, c_j)))$. 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 conclude that $\varphi' = \varphi$. Now, $\Phi(\overline{s}_i) = (u_1, c_1) \dots (u_j, c'_j) \dots (u_m, c_m)$ 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 $\mathcal{C}^{\#}_{\mathfrak{G}}$,

$$\alpha_0 \xrightarrow{\varphi_1, a_1} \alpha_1 \dots \alpha_{n-1} \xrightarrow{\varphi_n, a_n} \alpha_n$$

where each $\alpha_i \in \Gamma^*$.

Then, α_n is final if it is a sequence of completed nodes, which we write as $(\triangleright, \emptyset)(u_1, c_1) \dots (u_m, c_m)(\triangleleft, \emptyset)$. Then we claim that $\pi = u_1 \dots u_m$ 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 $i \in \{0, \dots, n\}$.

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 $\Phi(\overline{s}_{i-1}) = (u_1, c_1) \dots (u_m, c_m)$. Consider $\alpha_{i-1} \xrightarrow{\varphi_i, a_i} \alpha_i$. Then again we have two cases:

- either the transition executes event $(e, \beta'_1(u_j, c_j))$ enabled in $\alpha_{i-1} = \beta'_1(u_j, c_j)\beta'_2 = \beta'$ where $\beta'_1 = (\triangleright, \emptyset) Red_{\#}(\Psi((u_1, c_1) \dots (u_{j-1}, c_{j-1})))$ and $\beta'_2 = Red_{\#}(\Psi((u_{j+1}, c_{j+1}) \dots (u_m, c_m)))(\triangleleft, \emptyset)$.

- 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, $C_{i-1} \cap (E^{u_1 \dots u_j} \setminus E^{u_1 \dots u_{j-1}}) = \emptyset$. Thus, in this case, $c_j = \emptyset$.

Now, in both of the above cases, we claim that $(e, u_1 \dots u_j)$ is enabled in $\overline{s}_{i-1}$. Suppose not, choose a maximal event $(e', u_1 \dots u_{j'})$ which was not executed in $\overline{s}_{i-1}$, such that $(e', u_1 \dots u_{j'}) <^{\pi} (e, u_1 \dots u_j)$. 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 $C_i = C_{i-1} \uplus \{(e, u_1 \dots u_j)\}$. Then we can also observe that $\Phi(\overline{s}_i) = (u_1, c_1) \dots (u_j, c'_j) \dots (u_m, c_m)$ 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 6.16. *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 $\mathcal{L}_{tw}(\mathcal{C}^{\#}_{\mathfrak{G}})$.*

6.4.3 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 6.17. 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 6.18. *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 $Red(\alpha)$. Note that if $\alpha \xrightarrow{redn} \alpha'$, then $EProc(\alpha) = EProc(\alpha')$ and therefore, $EProc(\alpha) = EProc(Red(\alpha))$. In fact, from confluence, we derive some useful properties of the reduction operation,

(P1) $Red(\alpha_1\#\alpha_2) = Red(\alpha_1)\#Red(\alpha_2)$.

(P2) $Red(\alpha_1\alpha_2) = Red(Red(\alpha_1)\alpha_2) = Red(\alpha_1 Red(\alpha_2)) = Red(Red(\alpha_1)Red(\alpha_2))$

(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 $Red(\alpha) = \gamma_1(u,c)\gamma_2$ where $\gamma_1(u,c) = Red(\alpha_1(u,c))$ and $(u,c)\gamma_2 = Red((u,c)\alpha_2)$.

Proof. The first two properties are self-evident. For the third, using Remark 6.17 we deduce that (u,c) is not deleted during the reductions. Let $\gamma_1(u,c) = Red(\alpha_1(u,c))$ and $(u,c)\gamma_2 = Red((u,c)\alpha_2)$. Then applying Property (P2) twice on α, $Red(\alpha_1(u,c)\alpha_2) = Red(Red(\alpha_1(u,c))\alpha_2) = Red(\gamma_1(u,c)\alpha_2) = Red(\gamma_1 Red((u,c)\alpha_2)) = Red(\gamma_1(u,c)\gamma_2)$. Now since $\gamma_1(u,c)$ and $(u,c)\gamma_2$ are already in reduced form and (u,c) cannot be deleted in $Red(\alpha)$, we obtain $Red(\gamma_1(u,c)\gamma_2) = \gamma_1(u,c)\gamma_2$. □

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

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

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

Now, if we maintain the rest of the definition of a transition of $\mathcal{C}^{fin}_{\mathfrak{G}}$ to be the same as a transition of $\mathcal{C}^{\#}_{\mathfrak{G}}$, we can prove that $\mathcal{C}^{fin}_{\mathfrak{G}}$ 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 $\mathcal{C}^{fin}_{\mathfrak{G}}$, Properties (V1), (V2) and (V3) continue to hold.

Lemma 6.19. *If $\mathfrak{G}$ is locally synchronized, then $\mathcal{C}^{fin}_{\mathfrak{G}}$ as defined above is a finite MSC-ECA.*

Proof. We show that if $\mathfrak{G}$ is locally synchronized, then the number of states of $\mathcal{C}^{fin}_{\mathfrak{G}}$ is finite. For this, it is enough to show that the length of each reachable, completable state of $\mathcal{C}^{fin}_{\mathfrak{G}}$ 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 6.20. *Let $\alpha(u,c)\beta(u,c')\gamma$ be a valid completable state of $\mathcal{C}^{fin}_{\mathfrak{G}}$. Then if $(u,c)\beta$ is not completely executed or if # occurs in β, then $EProc((u,c')\gamma) \subsetneq EProc((u,c)\beta(u,c')\gamma)$.*

Proof. Let $e \in c$ such that $e \in E^u_p$ for some $p \in Proc$. Since the state $\alpha(u,c)\beta(u,c')\gamma$ is completable, for each occurence of # in β, there exists $u_1 \dots u_n \in V^*$ in G such that if we replace the # by $(u_1,\emptyset),\dots(u_n,\emptyset)$, 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 $p' = p_0 \to p_1 \to \dots \to p_n = p$ 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 $q \in (EProc((u,c)\beta') \setminus EProc((u,c')\gamma))$. We will now show that there is some good process $q \in \{p_0,\dots,p_n\}$.

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 6.21. *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 $2|Proc|$. From which we can conclude that we have a bound of $(2|Proc|)|V|$ on the number of extended nodes in a path. But

we know that each $\#$ or $P \subseteq Proc$ must have a node $u \in V$ next to it on the left, so we can conclude that the length of the path is $\mathcal{O}(|Proc||V|)$. Thus $\mathcal{C}_{\mathfrak{G}}^{fin}$ is finite. $\square$

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

$$\alpha \rightsquigarrow \beta \text{ if } \beta = Red(\alpha) \tag{6.8}$$

Now, we have the lemma,

Lemma 6.22. *$\rightsquigarrow$ is a bisimulation on abstract automata $\mathcal{C}_{\mathfrak{G}}^{\#}$ and $\mathcal{C}_{\mathfrak{G}}^{fin}$*

Proof. Let α be a state of $\mathcal{C}_{\mathfrak{G}}^{\#}$ and β a state of $\mathcal{C}_{\mathfrak{G}}^{fin}$ such that $\alpha \rightsquigarrow \beta$, i.e., $\beta = Red(\alpha)$.
($\Longrightarrow$) 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 $\mathcal{C}_{\mathfrak{G}}^{fin}$, where $\beta' = Red(\alpha')$. 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 $\beta_1 = Red(\alpha_1)$ and $\beta_2 = Red(\alpha_2)$. 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 $OProc(u) \cap EProc(\alpha_2) = \emptyset$, which implies that $OProc(u) \cap EProc(\beta_2) = \emptyset$ (since $\beta_2 = Red(\alpha_2)$). Notice that we also have for $i \in \{1, 2\}$, $\beta_i' = Red(\alpha_i')$ since $\beta_i = Red(\alpha_i)$.
  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, $\hat{\beta}(\hat{u}, \hat{c}) \preceq \beta_1' = Red(\alpha_1')$ 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 $\beta \xrightarrow{\varphi',a} \beta' = Red(\beta_1'(u, \{e\})\beta_2')$ which executes this enabled event in $\mathcal{C}_{\mathfrak{G}}^{fin}$.

Now, we show that the same guard is used, i.e., $\varphi' = \varphi$. For this, observe that $\varphi = \varphi^{edge}$ and $\varphi' = \varphi'^{edge}$ since there are no local-constraints. Now $\varphi^{edge} = (\mathrm{Y}^1_p \in I)$ for some $p \in Proc$ if and only if $e = min(E^u_p)$, $\alpha'_1 = \alpha_1 = \alpha''_1(u', c'')$, $EdgeC((u', u), p) = I$. 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 $\varphi'^{edge} = (\mathrm{Y}^1_p \in I)$.

Finally, we will be done with this case if we show that $Red(\alpha') = \beta'$. We have $\beta' = Red(\beta'_1(u, c')\beta'_2) = Red(Red(\alpha'_1)(u, c')Red(\alpha'_2))$. But by Property (P2) this is equal to $Red(\alpha'_1(u, c')\alpha'_2) = Red(\alpha')$ 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 $c' = c \cup \{e\}$ 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), $\beta = Red(\alpha) = \gamma_1(u, c)\gamma_2$, where $\gamma_1(u, c) = Red(\alpha_1(u, c))$ and $(u, c)\gamma_2 = Red((u, c)\alpha_2)$. 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 $\mathcal{C}^{fin}_{\mathfrak{G}}$. 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 $Red(\alpha') = \beta'$. Since $\alpha' = \alpha_1(u, c')\alpha_2$ is such that $c \subsetneq c' \subseteq E^u$, we have $Red(\alpha') = Red(\alpha_1(u, c')\alpha_2) = Red(\gamma_1(u, c')\gamma_2) = \beta'$. 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 $\mathcal{C}^{fin}_{\mathfrak{G}}$. More formally, we assume that $\beta \xrightarrow{\varphi,a} \beta'$ is a transition in $\mathcal{C}^{fin}_{\mathfrak{G}}$ and show that there is a transition $\alpha \xrightarrow{\varphi,a} \alpha'$ in $\mathcal{C}^{\#}_{\mathfrak{G}}$.

Let the transition in $\mathcal{C}^{fin}_{\mathfrak{G}}$ 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 $e \in E_p^u$, there exists an event $(e', \hat{\alpha_1}(v, c'))$ with $e' \in (E_p^v \setminus c')$ 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 $e' \in (E_p^v \setminus c')$ 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 6.23. $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{fin}) = \mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{\#})$

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 $\mathcal{C}_{\mathfrak{G}}^{fin}$. □

This completes the proof of the main theorem of this section.

Proof of Theorem 6.12. Given a locally synchronized TCMSG $\mathfrak{G}$, consider the finite MSC-ECA $\mathcal{C}_{\mathfrak{G}}^{fin}$. Then, by using the above corollary, Lemma 6.15 and Lemma 6.10, we conclude that $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{fin}) = \mathcal{L}_{tw}(\mathfrak{G})$. □

6.5 Solving the consistency problem

Now, we are in a position to solve the consistency problem introduced in the beginning of this chapter. Indeed, the hard work for this has already been done. From the set-up that we have established in the previous sections, we can easily deduce our result.

Theorem 6.24. *For a locally synchronized TCMSG $\mathfrak{G}$ and a TMPA $\mathcal{A}$, the consistency problem for $\mathfrak{G}$ and $\mathcal{A}$ is decidable, i.e., it is decidable to check if for all $T \in \mathcal{L}_{time}(\mathcal{A})$, there exists some $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$ such that T realizes $\mathfrak{M}$.*

Proof. The basic idea is to obtain the complement of the language accepted by the MSC-ECA $\mathcal{C}_{\mathfrak{G}}^{fin}$ by applying the determinization procedure for MSC-ECA. Recall that if $\mathcal{A}$ is a TMPA, then its global semantics is defined in terms of the timed automaton $\mathcal{B}_{\mathcal{A}}$. Indeed, we have the property that $T \in \mathcal{L}_{time}(\mathcal{A})$ if and only if there exists $\sigma \in \mathcal{L}_{tw}(\mathcal{B}_{\mathcal{A}})$ such that σ is timed linearization of T.

Using this, we may conclude that $\mathcal{A}$ is consistent with $\mathfrak{G}$ if and only if $\mathcal{L}_{tw}(\mathcal{B}_{\mathcal{A}}) \cap \overline{\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{fin})} = \emptyset$. Now, we compute the complement of $\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{fin})$. For this, we apply the universal automaton construction detailed in Section 6.2 to the MSC-ECA $\mathcal{C}_{\mathfrak{G}}^{fin}$.

Then, by choosing the accept states in the universal automaton to be all those states that do not contain a final state of $\mathcal{C}_{\mathfrak{G}}^{fin}$, we obtain by Corollary 6.8, an MSC-ECA $\mathcal{C}_1$ for the complement language.

It remains to define an equivalent finite timed automaton so that we can conclude that emptiness checking is decidable. Let $B \in \mathbb{N}_{>0}$ be such that each $\sigma \in \mathcal{C}_{\mathfrak{G}}^{fin}$ is B-bounded. As in the proof of Corollary 6.13, the existence of this bound follows from the fact that $\mathfrak{G}$ is locally synchronized.

Now, consider the timed automaton $\mathcal{B}_{\mathcal{C}_1}^B$ obtained by applying the construction described in Section 6.2.1 on the MSC-ECA $\mathcal{C}_1$. Further, we fix the set of final states to be the union of the states mentioned in Corollary 6.5(1) and (2). That is, all states of $\mathcal{B}_{\mathcal{C}_1}^B$ whose first components are final states of $\mathcal{C}_1$ are final states and $\perp$ is a final state as well.

Let σ be a timed word. Since $\mathcal{C}_1$ is complete (which follows from the universality property), if σ is not well-formed or B-bounded, by Corollary 6.5(2) (or rather Remark 6.2), it reaches $\perp$ and is therefore accepted. Otherwise, by Corollary 6.5(1) σ is accepted by $\mathcal{B}_{\mathcal{C}_1}^B$ if and only if it is accepted by $\mathcal{C}_1$.

Now observing that $\mathcal{C}_1$ accepts all timed words that are not well-formed or B-bounded, we conclude that $\mathcal{L}_{tw}(\mathcal{B}_{\mathcal{C}_1}^B) = \mathcal{L}_{tw}(\mathcal{C}_1)$. Thus, we have defined a finite timed automaton that recognizes $\overline{\mathcal{L}_{tw}(\mathcal{C}_{\mathfrak{G}}^{fin})}$ which means that checking its intersection with $\mathcal{L}_{tw}(\mathcal{B}_{\mathcal{A}})$ is decidable. Thus we conclude that it is decidable to check if $\mathcal{A}$ is consistent with $\mathfrak{G}$. □

6.6 Solving the coverage problem

Let $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ be a TCMSG and $\mathcal{A} = (\{\mathcal{A}_p\}_{p \in Proc}, \Delta, F, \mathcal{Z})$ a TMPA. We recall that the *coverage problem* for $\mathfrak{G}$ and $\mathcal{A}$ is to determine whether for each TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, there is a TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$.

Our strategy for the solution is as follows. Note that every TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$ is uniquely defined by the path followed in G to obtain it. We record the set of paths in G that can be witnessed by the TMPA $\mathcal{A}$ by synchronizing $\mathcal{A}$ with $\mathcal{C}_{\mathfrak{G}}^{fin}$. Comparing this set to the set of all paths in G, we obtain a solution to the coverage problem.

For recording a path, our strategy is to *emit* the sequence of nodes visited by the path. However, if a process, say p, does not participate in a node u but does participate in the next node v in the path, then by this strategy, we may emit v before u. Thus, we additionally need to handle the out of order emission of node labels. The problem is that, instead of a single node u, we could have a loop (which is still locally synchronized) in which p does not participate. In this case, it becomes very hard to recover the actual path traversed from the sequence of nodes emitted.

One way to get around this problem is by introducing a structural restriction on the TCMSG forbidding such behaviour. We propose a natural restriction that handles this in the following section.

6.6.1 Event-saturated TCMSGs

Definition 6.25. *A locally synchronized TCMSG $\mathfrak{G}$ is said to be* event-saturated *if in every node of $\mathfrak{G}$ there is an event present on each process.*

Example 32. We can observe that the TCMSG considered in Figure 6.1 is event-saturated as every process participates in each node of the graph.

Now we see why coverage is easier to establish for event-saturated TCMSGs. Intuitively, every move $\alpha \xrightarrow{\varphi,a} \alpha'$ in $\mathcal{C}^{\#}_{\mathfrak{G}}$ or $\mathcal{C}^{fin}_{\mathfrak{G}}$ between reachable and completable states, either executes an event in α or extends the current path by exactly one node. Formally,

Proposition 6.26. *Let $\mathfrak{G}$ be an event-saturated TCMSG and $\mathcal{C}^{\#}_{\mathfrak{G}}$ and $\mathcal{C}^{fin}_{\mathfrak{G}}$ be the associated MSC-ECA as defined in Sections 6.4.2 and 6.4.3, respectively. Then, all reachable and completable states of $\mathcal{C}^{\#}_{\mathfrak{G}}$ and $\mathcal{C}^{fin}_{\mathfrak{G}}$ are of the form $(\triangleright, \emptyset)\alpha\#(\triangleleft, \emptyset)$ or $(\triangleright, \emptyset)\alpha(\triangleleft, \emptyset)$ where $\alpha \in (ExtNodes)^*$. Further, if $\alpha = (u_1, c_1)\ldots(u_m, c_m)$, then $u_1 \ldots u_m$ is a path in G.*

Proof. For any node u, we have $OProc(u) = Proc$. Thus in any node insertion move $\alpha_1\#\alpha_2 \xrightarrow{u} \alpha'_1(u, \emptyset)\alpha'_2$, by Condition (I1) we infer that $EProc(\alpha_2) = \emptyset$ which implies that $\alpha_2 = (\triangleleft, \emptyset)$. In addition, from the fact that the state is completable we obtain $\alpha'_1 = \alpha_1$ and $\alpha'_2 = \#(\triangleleft, \emptyset)$ or $\alpha'_2 = (\triangleleft, \emptyset)$. Thus, from this and by Condition (I2), (I3) it follows that the node insertion extends the path with a single node. Thus, any move either executes an event in the current path or it is a node insertion which extends the path with a single node. Finally, when a reduction is applied in $\mathcal{C}^{fin}_{\mathfrak{G}}$, it always removes the leftmost node (which is not the endpoint $(\triangleright, \emptyset)$) in the current path. This follows from the definition of the reduction rules. Hence, we conclude that any completable state reached defines a path in G and the proposition follows. □

Remark 6.27. As observed earlier, each run of $\mathcal{C}^{\#}_{\mathfrak{G}}$ or $\mathcal{C}^{fin}_{\mathfrak{G}}$ defines a path through G. In this case, as each process occurs in each node of G, we can further infer that each process visits all the nodes in the path traced out by the run in the same order.

6.6.2 Coverage for event-saturated TCMSGs

Recall that our proof strategy is to record the paths that $\mathcal{A}$ can follow in $\mathfrak{G}$ by constructing a product of $\mathfrak{G}$ and $\mathcal{A}$. We enlarge the communication actions in *Act*

to include the set of nodes in $\mathfrak{G}$. Then, we build a product of the global timed automaton obtained from $\mathcal{A}$ and the timed automaton obtained from $\mathcal{C}^{fin}_{\mathfrak{G}}$ thus synchronizing the runs of $\mathcal{A}$ with the runs of $\mathcal{C}^{fin}_{\mathfrak{G}}$. The language of the resulting timed automaton would be the set of all runs of the TMPA that are consistent with some run of the TCMSG. Now, in this automaton, using our enlarged alphabet, we emit the nodes seen along these runs.

Finally, we use the region construction [5] to obtain an untimed regular language $Untime(\mathcal{L}_{tw}(\mathcal{B})) \subseteq (Act \cup Q)^*$. This language projected onto the alphabet Q precisely describes the set of all paths in $\mathfrak{G}$ that are covered by some run of $\mathcal{A}$.

The product construction Formally, we recall the product construction which gives a timed automaton $\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}}$ accepting the intersection of two timed languages $\mathcal{L}_{tw}(\mathcal{A})$ and $\mathcal{L}_{tw}(\mathfrak{G})$. We start with a bounded TMPA $\mathcal{A} = (\{\mathcal{A}_p\}_{p\in Proc}, \Delta, F, \mathcal{Z})$ and consider its global finite timed automaton $\mathcal{B}_{\mathcal{A}}$ as defined in Section 3.2, where $\overline{s}_{\mathcal{A}}$ denotes a state of $\mathcal{B}_{\mathcal{A}}$. Similarly from the locally synchronized TCMSG $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ we obtain the finite state timed automaton $\mathcal{B}_{\mathfrak{G}}$ by using Corollary 6.13. We let $\overline{s}_{\mathfrak{G}}$ be a state of $\mathcal{B}_{\mathfrak{G}}$. Without loss of generality, we assume that the set of clocks used by $\mathcal{B}_{\mathfrak{G}}$ and $\mathcal{B}_{\mathcal{A}}$ are disjoint.

Then, any state of the product automaton is of the form $(\overline{s}_{\mathcal{A}}, \overline{s}_{\mathfrak{G}})$. The set of clocks is the union of the set of clocks of $\mathcal{B}_{\mathcal{A}}$ and $\mathcal{B}_{\mathfrak{G}}$.

A transition is of the form $(\overline{s}_{\mathcal{A}}, \overline{s}_{\mathfrak{G}}) \xrightarrow{\varphi,a,R} (\overline{s}'_{\mathcal{A}}, \overline{s}'_{\mathfrak{G}})$ where

- $\varphi = \varphi_1 \wedge \varphi_2$ and $R = R_1 \cup R_2$, for some $\varphi_1 \in \mathrm{Form}(\mathcal{Z}_{\mathcal{A}})$, $\varphi_2 \in \mathrm{Form}(\mathcal{Z}_{\mathfrak{G}})$
- $(\overline{s}_{\mathcal{A}}, \varphi_1, a, R_1, \overline{s}'_{\mathcal{A}})$ is a global transition of $\mathcal{B}_{\mathcal{A}}$
- $(\overline{s}_{\mathfrak{G}}, \varphi_2, a, R_2, \overline{s}'_{\mathfrak{G}})$ is a global transition of $\mathcal{B}_{\mathfrak{G}}$

It then follows that

Lemma 6.28. *$\mathcal{L}_{tw}(\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}}) = \{\sigma \mid \sigma$ is a timed linearization of $T \in \mathcal{L}_{time}(\mathcal{A})$ and T realizes some $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})\}$*

Proof. This follows from Corollary 6.13 and the definition of acceptance of T by the TMPA $\mathcal{A}$. □

Now, we modify the construction by considering the alphabet $\Sigma = Act \times (Q \cup \diamond)$ where Q is the set of nodes of $\mathfrak{G}$ and $\diamond$ is an extra symbol. The set of states and clocks are the same as before. We redefine the transitions as follows: $(\overline{s}_{\mathcal{A}}, \overline{s}_{\mathfrak{G}}) \xrightarrow{\varphi,(a,b),R} (\overline{s}'_{\mathcal{A}}, \overline{s}'_{\mathfrak{G}})$ is a transition if $(\overline{s}_{\mathcal{A}}, \overline{s}_{\mathfrak{G}}) \xrightarrow{\varphi,a,R} (\overline{s}'_{\mathcal{A}}, \overline{s}'_{\mathfrak{G}})$ is a transition of the product automaton and

$$b = \begin{cases} q & \text{if } (\overline{s}_{\mathfrak{G}}, \varphi_2, a, R_2, \overline{s}'_{\mathfrak{G}}) \text{ and } \overline{s}'_{\mathfrak{G}} \text{ extends } \overline{s}_{\mathfrak{G}} \text{ by the node } q \in Q \\ \diamond & \text{otherwise} \end{cases}$$

The soundness of the above definition follows from Remark 6.27. Let us call this new timed automaton $\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}$. Then,

Lemma 6.29. *$\mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}) = \{\sigma \in \mathrm{TW}_{Act\times(Q\cup\diamond)} \mid \sigma = ((a_1,b_1),t_1)\dots((a_n,b_n),t_n)$ such that $(a_1,t_1)\dots(a_n,t_n)$ is a linearization of a TMSC $T \in \mathcal{L}_{time}(\mathcal{A})$ that realizes the TCMSC $\mathfrak{M}_{b_{i_1}\dots b_{i_m}} \in \mathcal{L}_{TC}(\mathfrak{G})$ where $b_{i_1}\dots b_{i_m}$ is the projection of $b_1\dots b_n$ onto $Q\}$.*

Proof. Consider the timed language obtained by projecting $\mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ onto its first component. That is, we define, $\mathcal{L}^1_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}) = \{(a_1,t_1)\dots(a_n,t_n) \in \mathrm{TW}_{Act} \mid$ there exists $\sigma = \sigma_1\dots\sigma_n \in \mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ such that $\forall i \in \{1,\dots,n\}$, $\sigma_i = ((a_i,b_i),t_i)\}$. Then, this language coincides with the language of $\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}}$ defined above, i.e, we have $\mathcal{L}^1_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}) = \mathcal{L}_{tw}(\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}})$. Thus, by Lemma 6.28, $(a_1,t_1)\dots(a_n,t_n) \in \mathcal{L}_{tw}(\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}})$ if and only if it is a timed linearization of some TMSC that realizes a TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$.

Now, consider $\mathcal{L}^2_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}) = \{b_1\dots b_n \in (Q\cup\{\diamond\})^* \mid$ there exists $\sigma = (\sigma_1\dots\sigma_n) \in \mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ such that $\forall i \in \{1,\dots,n\}$, $\sigma_i = ((a_i,b_i),t_i)\}$. Then, from Proposition 6.26 and the translation in Section 6.2.1, it follows that $\mathcal{L}^2_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ lists out the nodes of G in the order in which they are traversed by $\mathcal{B}^{prod}_{\mathfrak{G},\mathcal{A}}$. Therefore by projecting $\mathcal{L}^2_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ onto Q (i.e., by erasing $\diamond$), we obtain the actual path through G that corresponds to the run of $\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}$. In other words, the path $b_{i_1}\dots b_{i_m}$ obtained by projecting $b_1\dots b_n$ to Q is exactly the path that generates the TCMSC $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, which completes the proof. □

We define $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}) = \{b_1\dots b_n \in Q^* \mid$ there exists $w \in \mathit{Untime}(\mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}}))$ such that w projected on its second component and projected onto the alphabet Q gives $b_1\dots b_n\}$.

To check coverage, we just need to verify that the *node language* of G, $\mathcal{L}_Q(\mathfrak{G}) = \{q_0q_1\dots q_n \in Q^* \mid q_0 \to q_1 \to \cdots \to q_n$ is a run$\}$, is included in $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$. This would imply that for every path π through $\mathfrak{G}$, the TCMSC $\mathfrak{M}_\pi$ is realized by some TMSC in $\mathcal{L}_{time}(\mathcal{A})$.

Lemma 6.30. *If $\mathcal{L}_Q(\mathfrak{G}) \subseteq \mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$, then for all $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, there exists $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$.*

Proof. $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$ implies that $\mathfrak{M} = \mathfrak{M}_\pi$ for some path $\pi \in \mathcal{L}_Q(\mathfrak{G})$. But then $\pi \in \mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ which implies by definition that there exists $\sigma \in \mathcal{L}_{tw}(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ such that $\mathit{Untime}(\sigma)$ projected onto Q is π. But then by Lemma 6.29, we are done since σ projected onto the first component gives us the witness. □

For the converse, however, there is a slight complication. Some paths in $\mathfrak{G}$ may define TCMSCs that cannot be realized, because of self-contradictory timing constraints. Therefore, it is not enough to directly compare $\mathcal{L}_Q(\mathfrak{G})$ with $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$. The solution is straightforward: we start with the trivial automaton $\mathcal{A}_U$ that

recognizes Act^*, which can be regarded as a degenerate timed automaton with no timing constraints. To $\mathcal{A}_U$, we apply the same construction as we have done for $\mathcal{A}$. The resulting timed automaton $\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}_U}$ will mark out all paths π through $\mathfrak{G}$ for which $\mathfrak{M}_\pi$ can be realized by some TMSC. Hence, checking that every realisable path is witnessed by $\mathcal{A}$ amounts to checking that $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}_U})$ is included in $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$.

Lemma 6.31. *$\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}_U}) \subseteq \mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ if and only if for all $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$ there exists $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$.*

Since both $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}_U})$ and $\mathcal{L}^2_Q(\mathcal{B}^{cov}_{\mathfrak{G},\mathcal{A}})$ are regular languages, the result follows.

Theorem 6.32. *For an event-saturated locally synchronized TCMSG $\mathfrak{G}$ and a TMPA $\mathcal{A}$, the coverage problem for $\mathfrak{G}$ and $\mathcal{A}$ is decidable, i.e, it is decidable to check if for all $\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})$, there exists $T \in \mathcal{L}_{time}(\mathcal{A})$ such that T realizes $\mathfrak{M}$.*

Part II

Changing the model

7

Distributed Timed Automata with Independently Evolving Clocks

In the previous chapters, while considering systems with time and concurrency, we have always restricted ourselves to a uniform global time over all the constituent processes. In some sense, we have concentrated on the distributed behaviour of the systems and then incorporated timing as an additional feature.

In this chapter, we would like to concentrate on the distributed behaviour of systems where time itself plays a more important, synchronizing role. In other words, we want to focus on distributed systems where each constituent process can have a different time evolution. This allows us to ask questions like, can we use clocks as a synchronization tool between processes, what global behaviours can such a system exhibit or deny and so on.

As a first step towards this goal, we introduce a model of a system that allows different constituent processes to evolve at different time rates. In order to focus our attention on time as a tool to enforce/exhibit synchronization, we only consider the untimed global (sequential) behaviours. As mentioned in the introduction, this differentiates our approach from the earlier ones which consider timed semantics. Indeed we avoid giving a timed semantics, as it would involve introducing global time in some way. However, in the conclusion section, we will give pointers to future extensions where we might consider partial order (but still untimed) behaviours as well.

7.1 The model

Let us recall the notion of timed automata [5] as defined in the preliminaries chapter. These will constitute the building blocks of our distributed timed automata. A *finite timed automaton* is a tuple $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F)$ where S is a finite set of *states*, Σ is the alphabet of *actions*, $\mathcal{Z}$ is a finite set of *clocks*, $\delta \subseteq S \times \Sigma_\varepsilon \times \mathrm{Form}(\mathcal{Z}) \times 2^{\mathcal{Z}} \times S$ is the finite set of *transitions*, $Inv : S \to (\mathcal{Z})$ associates with each state an *invariant*, $\iota \in S$ is the *initial state*, and $F \subseteq S$ is the

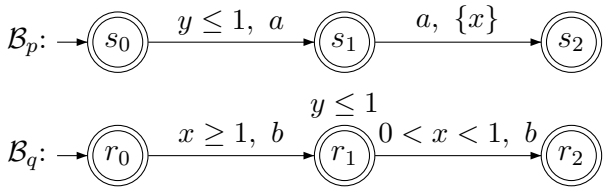


Figure 7.1: A distributed timed automaton over $\{p, q\}$

set of *final states*. We let $\mathit{Reset}(\mathcal{B}) = \{x \in \mathcal{Z} \mid \text{there is } (s, \varphi, a, R, s') \in \delta \text{ such that } x \in R\}$ be the set of clocks that might be reset in $\mathcal{B}$. Without loss of generality, we will assume in this chapter that $\mathit{Inv}(\iota)$ is satisfied by the clock valuation over $\mathcal{Z}$ that maps each clock to 0.

We will now extend the above definition to a distributed setting. First, we fix a non-empty finite set *Proc* of processes (unless otherwise stated). For a tuple t that is indexed by *Proc*, t_p refers to the projection of t onto $p \in \mathit{Proc}$.

Definition 7.1. *A* distributed timed automaton (DTA) *over the set of processes Proc is a structure* $\mathcal{D} = ((\mathcal{B}_p)_{p \in \mathit{Proc}}, \pi)$ *where* $\mathcal{B}_p = (S_p, \Sigma_p, \mathcal{Z}_p, \delta_p, \mathit{Inv}_p, \iota_p, F_p)$ *are finite timed automata such that the alphabets* Σ_p *are pairwise disjoint and* π *is a (total) map from* $\bigcup_{p \in \mathit{Proc}} \mathcal{Z}_p$ *to Proc such that, for each* $p \in \mathit{Proc}$*, we have* $\mathit{Reset}(\mathcal{B}_p) \subseteq \pi^{-1}(p) \subseteq \mathcal{Z}_p$.

Note that $\mathcal{Z}_p$ refers to the set of clocks that might occur in the timed automaton $\mathcal{B}_p$, either as clock guard or reset. The same clock may occur in both $\mathcal{Z}_p$ and $\mathcal{Z}_q$, since it may be read as a guard in both processes. However, any clock evolves according to the time evolution of some particular process. This clock is then said to *belong* to that process, and the *owner* map, π, formalizes this in the above definition. This will become more clear when we describe the formal semantics later in this section. Finally, we assume that a clock can only be reset by the process it belongs to.

Example 33. Suppose $\mathit{Proc} = \{p, q\}$. Consider the DTA $\mathcal{D}$ as given by Figure 7.1. It consists of two timed automata, $\mathcal{B}_p$ and $\mathcal{B}_q$ with $\mathcal{Z}_p = \mathcal{Z}_q = \{x, y\}$. In both automata, we suppose all states to be final. Moreover, the owner map π assigns clock x to process p and clock y to process q. Note that $\mathit{Reset}(\mathcal{B}_p) = \{x\}$ and $\mathit{Reset}(\mathcal{B}_q) = \emptyset$. Before we define the semantics of $\mathcal{D}$ formally and in a slightly more general setting, let us give some intuitions on the behavior of $\mathcal{D}$. If both clocks are completely synchronized, i.e., they follow the same local clock rate, then our model corresponds to a standard network of timed automata [18]. For example, we might execute a within one time unit, and at time 1, execute b, ending up in the global state (s_1, r_1) and clock valuation $\nu(x) = \nu(y) = 1$. If we now wanted to perform a further b, this should happen instantaneously. But this also requires a reset of x in the automaton $\mathcal{B}_p$ and, in particular, a time elapse greater than zero,

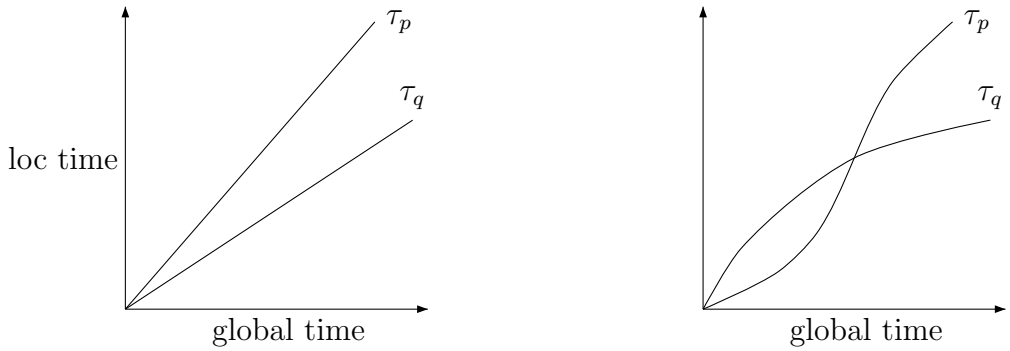


Figure 7.2: Examples of local time rate functions

violating the invariant at the local state r_1. Thus, the word *abab* will not be in the semantics that we associate with $\mathcal{D}$ wrt. synchronized local-time evolution. Now suppose clock y runs slower than clock x. Then, having executed *ab*, we might safely execute a further a while resetting x and, then, let some time elapse without violating the invariant. Thus, *abab* will be contained in the *existential* semantics, as there are local time evolutions that allow for the execution of this word. Observe that a and aa are the only sequences that can be executed no matter what the relative time speeds are: the guard $y \leq 1$ is always satisfied for a while. But we cannot guarantee that the guard $x \geq 1$ and the invariant $y \leq 1$ are satisfied at the same time, which prevents a word containing b from being in the *universal* semantics of $\mathcal{D}$.

The semantics The semantics of a DTA depends on the (possibly dynamically changing) time rates at the processes. To model this, we assume that these rates depend on some absolute time, i.e., they are given by a tuple $\tau = (\tau_p)_{p \in Proc}$ of functions $\tau_p : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. Thus, each local time function maps every point in global time to some local time instant. Then, we require (justifiably) that these functions are continuous, strictly increasing, and divergent. Further, they satisfy $\tau_p(0) = 0$ for all $p \in Proc$. The set of all these tuples τ is denoted by $Rates$. We might also consider τ as a mapping $\mathbb{R}_{\geq 0} \to (\mathbb{R}_{\geq 0})^{Proc}$ so that, for $t \in \mathbb{R}_{\geq 0}$, $\tau(t)$ denotes the tuple $(\tau_p(t))_{p \in Proc}$.

By superimposing the local time rate maps for each process on the same graph, we can represent τ pictorially, as in Figure 7.2. Thus, in the first picture, clocks on process p evolve steadily faster than clocks on process q. Whereas, in the second picture, the clocks on process q are initially faster than clocks on process p but start to lag behind them after some time.

Now, a distributed system can usually be described by an asynchronous product of automata. In the case of DTA, the semantics can be defined using such a product and a mapping that assigns any clock to its owner process. For this, we start by introducing the following more general model, with a unified state space, for which

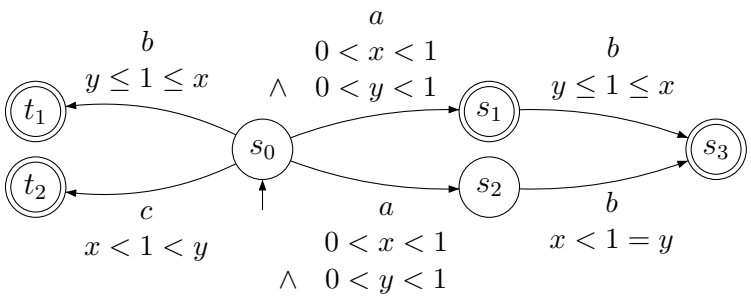


Figure 7.3: An icTA $\mathcal{B}$ with independent clocks x and y

it will be easier to define the semantics.

Definition 7.2. *A* timed automaton with independently evolving clocks (icTA) *over Proc is a tuple* $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ *where* $(S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F)$ *is a timed automaton and* $\pi : \mathcal{Z} \to Proc$ *maps each clock to a process.*

Now, we would like to define a run of an icTA. Intuitively, this is done in the same spirit as a run of a timed automaton over a timed word except for one difference. The time evolution, though according to absolute time, is perceived by each process as its *local time* evolution. Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ be an icTA. Then, given a clock valuation $\nu : \mathcal{Z} \to \mathbb{R}$ and a tuple $t \in \mathbb{R}^{Proc}$, the valuation $\nu + t$ is defined by $(\nu + t)(x) = \nu(x) + t_{\pi(x)}$ for all $x \in \mathcal{Z}$.

Thus, for $\tau \in Rates$, we define a τ*-run* of $\mathcal{B}$ as a sequence

$$(s_0, \nu_0) \xrightarrow{a_1,t_1} (s_1, \nu_1) \xrightarrow{a_2,t_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n,t_n} (s_n, \nu_n)$$

where $n \geq 0$, $s_i \in S$, $a_i \in \Sigma_\varepsilon$, and $(t_i)_{1 \leq i \leq n}$ is a non-decreasing sequence of values from $\mathbb{R}_{\geq 0}$. Further, $\nu_i : \mathcal{Z} \to \mathbb{R}_{\geq 0}$ with $\nu_0(x) = 0$ for all $x \in \mathcal{Z}$. Finally, for all $i \in \{1, \ldots, n\}$, there are $\varphi_i \in \mathrm{Form}(\mathcal{Z})$ and $R_i \subseteq \mathcal{Z}$ such that the following conditions hold:

$$(s_{i-1}, \varphi_i, a_i, R_i, s_i) \in \delta \tag{7.1}$$

$$\nu_{i-1} + \tau(t') - \tau(t_{i-1}) \models Inv(s_{i-1}) \qquad \text{for each } t' \in [t_{i-1}, t_i] \tag{7.2}$$

$$\nu_{i-1} + \tau(t_i) - \tau(t_{i-1}) \models \varphi_i \tag{7.3}$$

$$\nu_i = (\nu_{i-1} + \tau(t_i) - \tau(t_{i-1}))[R_i \to 0] \tag{7.4}$$

$$\nu_i \models Inv(s_i) \tag{7.5}$$

In this case, we write $(\mathcal{B}, \tau) : s_0 \xrightarrow{a_1 \cdot \ldots \cdot a_n} s_n$ or also $(\mathcal{B}, \tau) : s_0 \xrightarrow{a_1 \cdot \ldots \cdot a_i} s_i \xrightarrow{a_{i+1} \cdot \ldots \cdot a_n} s_n$ to abstract from the time instances. The latter thus denotes that $\mathcal{B}$ can, reading w, go from s_0 via s_i to s_n, while respecting the local-time rates τ.

Definition 7.3. *Let* $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ *be an* icTA *and* $\tau \in Rates$*. The* language *of* $\mathcal{B}$ *wrt.* τ*, denoted by* $\mathcal{L}(\mathcal{B}, \tau)$*, is the set of words* $w \in \Sigma^*$ *such that* $(\mathcal{B}, \tau) : \iota \xrightarrow{w} s$ *for some* $s \in F$*. Moreover, we define* $\mathcal{L}_\exists(\mathcal{B}) = \bigcup_{\tau \in Rates} \mathcal{L}(\mathcal{B}, \tau)$ *to be the* existential semantics *and* $\mathcal{L}_\forall(\mathcal{B}) = \bigcap_{\tau \in Rates} \mathcal{L}(\mathcal{B}, \tau)$ *to be the* universal semantics *of* $\mathcal{B}$*.*

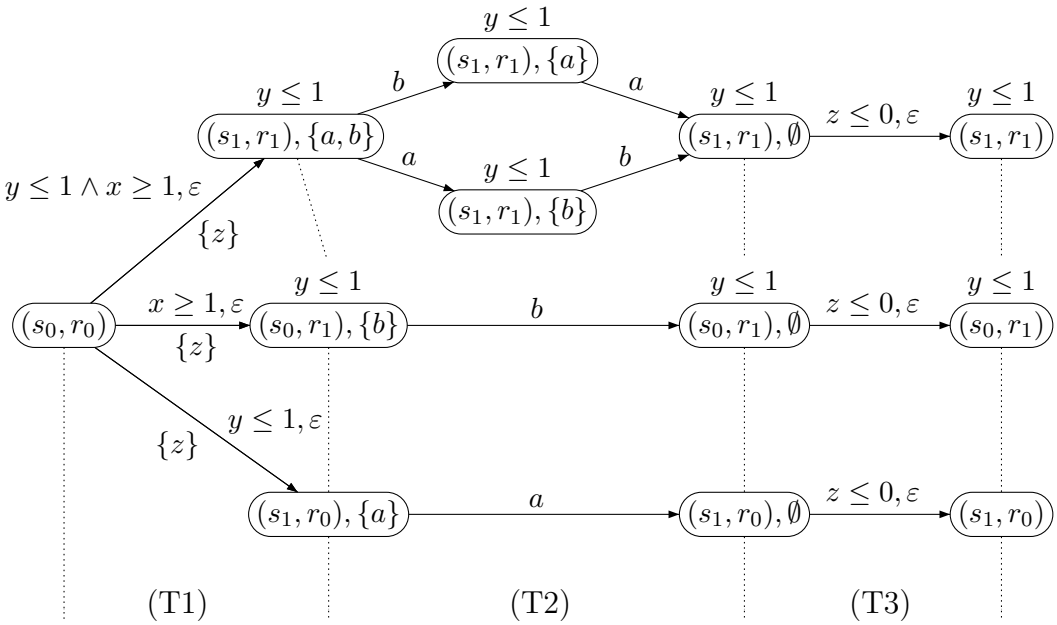


Figure 7.4: Part of the icTA $\mathcal{B}_\mathcal{D}$ for the DTA $\mathcal{D}$ from Figure 7.1

If $|Proc| = 1$, then an icTA $\mathcal{B}$ actually reduces to an ordinary timed automaton and we have $\mathcal{L}_\forall(\mathcal{B}) = \mathcal{L}(\mathcal{B}, \tau) = \mathcal{L}_\exists(\mathcal{B})$ for any $\tau \in Rates$. Moreover, if $|Proc| \ge 1$ and $\tau \in Rates$ exhibits, for all $p \in Proc$, the same local time evolution, then $\mathcal{L}(\mathcal{B}, \tau)$ is the untimed language of $\mathcal{B}$ considered as an ordinary timed automaton.

Example 34. Consider a sample icTA $\mathcal{B}$ over the set of processes $\{p, q\}$ and $\Sigma = \{a, b, c\}$ as depicted in Figure 7.3. Assuming $\pi(x) = p$ and $\pi(y) = q$, we have $\mathcal{L}(\mathcal{B}, \mathrm{id}) = \{a, ab, b\}$, where id_p is the identity on $\mathbb{R}_{\ge 0}$ for all $p \in Proc$ (i.e., id models synchronization of any process with the absolute time). And we observe that $\mathcal{L}_\forall(\mathcal{B}) = \{a, ab\}$, $\mathcal{L}_\exists(\mathcal{B}) = \{a, ab, b, c\}$.

Now, we define the semantics of a DTA in terms of an associated icTA. This is obtained by taking a product of the components of the DTA which is slightly more complicated than a direct asynchronous product, in the sense that it simulates the simultaneous firing of independent actions (as in the DTA).

Thus, given a DTA $\mathcal{D} = ((\mathcal{B}_p)_{p \in Proc}, \pi)$ with $\mathcal{B}_p = (S_p, \Sigma_p, \mathcal{Z}_p, \delta_p, Inv_p, \iota_p, F_p)$, we associate an icTA, $\mathcal{B}_\mathcal{D} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ as defined below. The set of states of S consists of the product of the set of all local states S_p. In addition, we add states of the form (s, A) where s is a product of local states and $A \subseteq \Sigma$ is a set of actions. Intuitively, from a given configuration of a DTA, we guess the set of processes $P \subseteq Proc$, that may decide to fire an action (independently) and collect these actions into the set A. Now, we use a clock invariant to ensure that the icTA can simulate all these actions, in any order, without time elapsing in between. Formally, $S = (\prod_{p \in Proc} S_p) \cup \big((\prod_{p \in Proc} S_p) \times 2^\Sigma\big)$.

Now the alphabet is the disjoint union $\Sigma = \biguplus_{p \in Proc} \Sigma_p$, but for the clocks, we add one extra clock $\mathcal{Z} = \bigcup_{p \in Proc} \mathcal{Z}_p \cup \{z\}$ where $z \notin \mathcal{Z}_p$ for all $p \in Proc$. We will use this clock to ensure that some transitions occur instantaneously, i.e, without any time elapse. Thus, with our assumption stating that if time elapses, then it must elapse in all processes, we can assume z to belong to any process, i.e, $\pi(z)$ to be arbitrary.

We define the state invariants as $Inv(s) = \bigwedge_{p \in Proc} Inv_p(s_p)$, $Inv(s, A) = Inv(s)$. Also, $\iota = (\iota_p)_{p \in Proc}$, and $F = \prod_{p \in Proc} F_p$. Then, the transitions in $\mathcal{B}_\mathcal{D}$ are of three types:

(T1) The first type is an ε-move which guesses the set of processes of the DTA that will move next and the transitions that each of them would perform. In addition it checks the guard that each of them must satisfy and resets the clocks as well. Thus, $(s, \varphi, \varepsilon, R, (s', A)) \in \delta$ if there exists a set of processes $P \subseteq Proc$ and transitions $(s_p, \varphi_p, a_p, R_p, \widetilde{s}_p) \in \delta_p$ for each $p \in P$ such that,

- $s = (s_p)_{p \in Proc} \in S$
- $\varphi = \bigwedge_{p \in P} \varphi_p$,
- $R = \bigcup_{p \in P} R_p \cup \{z\}$,
- $s'_p = \widetilde{s}_p$ for $p \in P$, $s'_q = s_q$ for $q \notin P$.
- $A = \{a_p \mid p \in P\} \setminus \{\varepsilon\}$

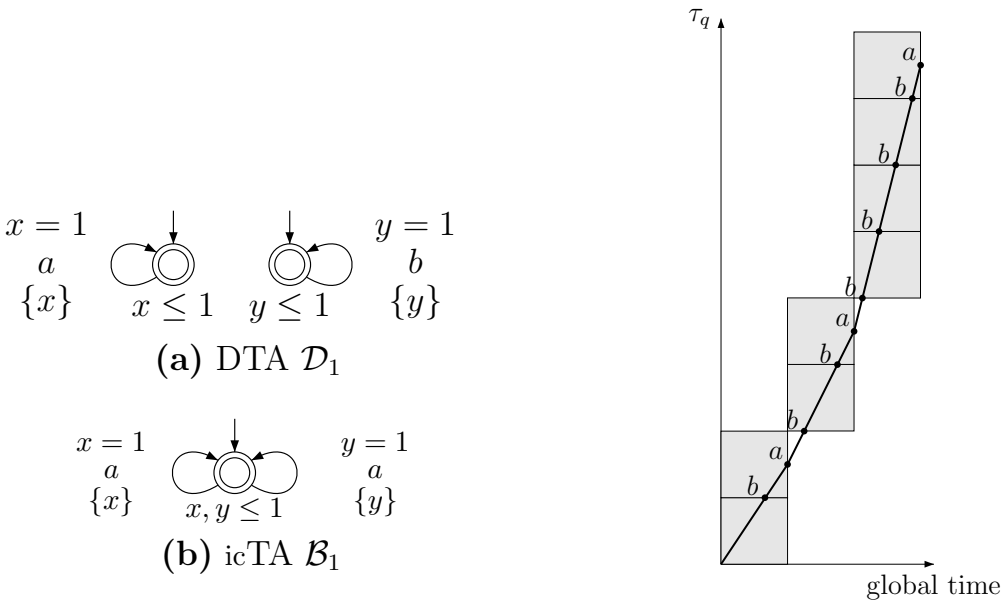
(T2) This move performs an action from its guessed set A and then removes it, i.e, $((s, A), \text{true}, a, \emptyset, (s, (A \setminus \{a\}))) \in \delta$ for all $(s, A) \in S$ and $a \in A$

(T3) This move allows to continue the simulation if all the guessed actions have been performed without any time-elapse, i.e, $((s, \emptyset), z \leq 0, \varepsilon, \emptyset, s) \in \delta$ for all $s \in S$.

This completes our definition of the icTA $\mathcal{B}_\mathcal{D}$ associated to a DTA $\mathcal{D}$.

Example 35. The Figure 7.4 illustrates how this construction works on the DTA $\mathcal{D}$ from Figure 7.1. The picture illustrates how each move of $\mathcal{D}$ is in fact split into three phases. In the first phase, from a product of local states of the DTA, transition T1 chooses the set of processes that may fire next. Then in the second phase, depending on this set we have a sequence of T2 transitions. In the final phase we have a T3 transition which results in a state which is again just a product of local states of $\mathcal{D}$. Indeed, all these three phases occur without time elapse since clock z is reset at T1 and checked to be zero at T3.

Now we can define,



(a) DTA $\mathcal{D}_1$

(b) icTA $\mathcal{B}_1$

Figure 7.5: An example of "weird" behaviour

Definition 7.4. *For a DTA $\mathcal{D}$ and $\tau \in Rates$, we set $\mathcal{L}(\mathcal{D}, \tau) = \mathcal{L}(\mathcal{B}_\mathcal{D}, \tau)$ to be the* language *of $\mathcal{D}$ wrt. τ, and we define $\mathcal{L}_\exists(\mathcal{D}) = \mathcal{L}_\exists(\mathcal{B}_\mathcal{D})$ as well as $\mathcal{L}_\forall(\mathcal{D}) = \mathcal{L}_\forall(\mathcal{B}_\mathcal{D})$ to obtain its existential and universal semantics, respectively.*

Example 36. For the DTA $\mathcal{D}$ that is given in Figure 7.1, we can now formalize what we had described intuitively: $\mathcal{L}(\mathcal{D}, \mathrm{id}) = \mathit{Pref}(\{aab, aba, baa\})$, $\mathcal{L}_\exists(\mathcal{D}) = \mathit{Pref}(\{aab, abab, baab\})$, and $\mathcal{L}_\forall(\mathcal{D}) = \mathit{Pref}(\{aa\})$ where, for $L \subseteq \Sigma^*$, $\mathit{Pref}(L)$ is the set of prefixes of words in L.

It is worthwhile to observe that $\mathcal{L}(\mathcal{D}, \tau)$ can, in general, have bizarre (non-regular) behavior, if τ is itself a "weird" function. This is one more reason to look at the existential and universal semantics. Let us quantify this with an example.

Example 37. Consider the simple DTA $\mathcal{D}_1$ in Figure 7.5 over $\mathit{Proc} = \{p, q\}$, where $\Sigma = \{a, b\}$, $\pi(x) = p$ and $\pi(y) = q$. The icTA $\mathcal{B}_1$ depicted in Figure 7.5 is a simplified version of $\mathcal{B}_{\mathcal{D}_1}$ where all the intermediate states and transitions have been removed. Indeed $\mathcal{L}(\mathcal{B}_1, \tau) = \mathcal{L}(\mathcal{B}_{\mathcal{D}_1}, \tau) = \mathcal{L}(\mathcal{D}_1, \tau)$ for any $\tau \in \mathit{Rates}$. Now, let $\tau = (\mathrm{id}_p, \tau_q)$, where τ_q is any continuous, strictly increasing function such that $\tau_q(0) = 0$ and $\tau_q(n) = 2^n - 0.5$ for any $n \geq 1$. This is seen in the adjoining graph in Figure 7.5. Then, an a occurs at every local time unit of p (which is the same as a unit of global time), and a b occurs at every local time unit of q. Thus, $\mathcal{L}(\mathcal{B}_1, \tau) = \mathcal{L}(\mathcal{D}_1, \tau)$ is the set of finite prefixes of the infinite word $bab^2ab^4ab^8ab^{16}a\ldots$, which is not a regular language.

We end this section by noting that icTAs, in fact, present a unified framework for many variants of a shared-memory model and their semantics. For instance, in the spirit of *asynchronous automata* [58], we could have considered a distributed timed automaton to be a tuple $\big((S_p)_p, (\Sigma_p)_p, \mathcal{Z}, (\delta_a)_a, (\mathit{Inv}_p)_p, \iota, (F_p)_p, \pi\big)$, where a

ranges over $\Sigma = \bigcup_{p \in Proc} \Sigma_p$, $\iota \in \prod_{p \in Proc} S_p$, and $\pi : \mathcal{Z} \to Proc$. This models a shared-memory system: executing an action $a \in \Sigma$ does not only affect one single process but rather involves each process from $proc(a) = \{p \in Proc \mid a \in \Sigma_p\}$. Therefore,

$$\delta_a \subseteq S_a \times \mathrm{Form}(\mathcal{Z}) \times 2^{\mathcal{Z}_a} \times S_a$$

where $S_a = \prod_{p \in proc(a)} S_p$ and $\mathcal{Z}_a = \bigcup_{p \in proc(a)} \pi^{-1}(p)$. The model from [11] corresponds to such an asynchronous automaton except for two differences: (1) clocks are local in the sense that they can only be read by those processes to which they belong, and (2) each process comes with a distinguished clock that is never reset; a synchronizing transition from δ_a can then be performed only if the special clocks that are associated with processes from $proc(a)$ exhibit the same value.

7.2 The existential semantics and the region abstraction

Given an icTA $\mathcal{B}$ (which might arise from some DTA $\mathcal{D}$) and a set Bad of undesired behaviors, it is natural to ask if $\mathcal{B}$ is robust against the (unknown) relative clock speeds and faithfully avoids executing action sequences from Bad. This corresponds to checking if $\mathcal{L}_\exists(\mathcal{B}) \cap Bad = \emptyset$. In this section, we show that this question is indeed decidable, given that Bad is a regular language. To this aim, we define a partitioning of clock valuations into finitely many equivalence classes and generalize the well-known region construction for timed automata [5].

But first, we give an alternate view of the icTA semantics as an infinite-state transition system. Given an icTA $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$, we associate $TS(\mathcal{B})$ as follows: A state of $TS(\mathcal{B})$ is a tuple (s, ν), where $s \in S$ and $\nu : \mathcal{Z} \to \mathbb{R}_{\geq 0}$. Then, for $a \in \Sigma_\varepsilon$, $(s, \nu) \xrightarrow{a} (s', \nu')$ is a transition if there exist $t \in \mathbb{R}_{\geq 0}$, $\tau \in Rates$, $\varphi \in \mathrm{Form}(\mathcal{Z})$ and $R \subseteq \mathcal{Z}$ such that:

$$(s, \varphi, a, R, s') \in \delta \tag{7.6}$$
$$\nu + \tau(t') \models Inv(s) \qquad \text{for each } t' \in [0, t] \tag{7.7}$$
$$\nu + \tau(t) \models \varphi \tag{7.8}$$
$$\nu' = (\nu + \tau(t))[R \to 0] \tag{7.9}$$
$$\nu' \models Inv(s') \tag{7.10}$$

The initial state is (s_0, ν_0) with $s_0 = \iota$ and $\nu_0(x) = 0$ for all $x \in \mathcal{Z}$. A state (s, ν) is final if $s \in F$. A run of $TS(\mathcal{B})$ on $w = a_1 \dots a_n \in \Sigma^*$ is a sequence of transitions,

$$(s_0, \nu_0) \xrightarrow{a_1} (s_1, \nu_1) \xrightarrow{a_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n} (s_n, \nu_n)$$

where $n \geq 0$. It is accepting if $s_n \in F$ and in this case we say $w \in \mathcal{L}(TS(\mathcal{B}))$.

Proposition 7.5. $\mathcal{L}(\mathit{TS}(\mathcal{B})) = \mathcal{L}_\exists(\mathcal{B})$.

Proof. Consider $w \in \mathcal{L}_\exists(\mathcal{B})$. Then $w \in \mathcal{L}(\mathcal{B}, \tau)$ for some τ and we find an accepting τ-run of $\mathcal{B}$:

$$(s_0, \nu_0) \xrightarrow{a_1,t_1} (s_1, \nu_1) \xrightarrow{a_2,t_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n,t_n} (s_n, \nu_n)$$

with $a_i \in \Sigma_\varepsilon$, $w = a_1 \dots a_n$ and such that (7.1–7.5) hold for some $\varphi_i \in \mathrm{Form}(\mathcal{Z})$ and $R_i \subseteq \mathcal{Z}$. We show that, abstracting away from the t_i's, we obtain a run of $\mathit{TS}(\mathcal{B})$. For $1 \leq i \leq n$, we define $\hat{t}_i = t_i - t_{i-1}$ (with $t_0 = 0$) and τ_i by $\tau_i(t) = \tau(t_{i-1} + t) - \tau(t_{i-1})$. From (7.2–7.4), we obtain

$$\nu_{i-1} + \tau_i(t') \models \mathit{Inv}(s_{i-1}) \qquad \text{for each } t' \in [0, \hat{t}_i] \tag{7.11}$$

$$\nu_{i-1} + \tau_i(\hat{t}_i) \models \varphi_i \tag{7.12}$$

$$\nu_i = (\nu_{i-1} + \tau_i(\hat{t}_i))[R_i \to 0] \tag{7.13}$$

Therefore, $(s_0, \nu_0) \xrightarrow{a_1} (s_1, \nu_1) \xrightarrow{a_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n} (s_n, \nu_n)$ is an accepting run of $\mathit{TS}(\mathcal{B})$ for w.

Conversely, let $w = a_1 \dots a_n \in \mathcal{L}(\mathit{TS}(\mathcal{B}))$ and let

$$(s_0, \nu_0) \xrightarrow{a_1} (s_1, \nu_1) \xrightarrow{a_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n} (s_n, \nu_n)$$

be an accepting run of $\mathit{TS}(\mathcal{B})$ for w. By definition, for each $1 \leq i \leq n$, we find $\hat{t}_i \geq 0$, τ_i, φ_i and R_i such that (7.1,7.11–7.13,7.5) are satisfied. We define now by induction the non-decreasing sequence $(t_i)_{0 \leq i \leq n}$ by $t_0 = 0$ and $t_i = t_{i-1} + \hat{t}_i$ for $1 \leq i \leq n$. We also define τ in order to obtain a τ-run of $\mathcal{B}$: for $1 \leq i \leq n$ and $t \in [t_{i-1}, t_i]$, we let $\tau(t) = \tau(t_{i-1}) + \tau_i(t - t_{i-1})$; and for $t \geq t_n$, we let $\tau(t) = \tau(t_n) + \mathrm{id}(t - t_n)$. Then, we can easily check using (7.11–7.13) that (7.2–7.4) are satisfied. Therefore,

$$(s_0, \nu_0) \xrightarrow{a_1,t_1} (s_1, \nu_1) \xrightarrow{a_2,t_2} (s_2, \nu_2) \cdots (s_{n-1}, \nu_{n-1}) \xrightarrow{a_n,t_n} (s_n, \nu_n)$$

is an accepting τ-run of $\mathcal{B}$.

□

In order to prove that $\mathcal{L}_\exists(\mathcal{B})$ is regular, we define on $\mathit{TS}(\mathcal{B})$ a bisimulation of finite index which preserves final states. In this way, we obtain as a quotient a finite automaton accepting $\mathcal{L}(\mathit{TS}(\mathcal{B})) = \mathcal{L}_\exists(\mathcal{B})$. As one may expect, this bisimulation is based on clock regions that we define below.

For each clock $x \in \mathcal{Z}$, let C_x be the largest constant clock x is compared with in guards or invariants. Let $p \in \mathit{Proc}$. As before, $\pi^{-1}(p) = \{z \in \mathcal{Z} \mid \pi(z) = p\}$ denotes the set of clocks owned by p. Given a clock valuation ν over $\mathcal{Z}$, define its p-restriction $\nu_p : \pi^{-1}(p) \to \mathbb{R}_{\geq 0}$ by $\nu_p(x) = \nu(x)$ for all $x \in \pi^{-1}(p)$. Then, from the classical region construction for timed automata, we obtain a notion of equivalence $\sim_p$ between two such valuations. That is, we say that $\nu_p \sim_p \nu'_p$ if the following hold:

1. for each $x \in \pi^{-1}(p)$, $\nu_p(x) > C_x$ if and only if $\nu'_p(x) > C_x$,
2. for each $x \in \pi^{-1}(p)$, $\nu_p(x) \leq C_x$ implies both $\lfloor \nu_p(x) \rfloor = \lfloor \nu'_p(x) \rfloor$ and $\mathit{fract}(\nu_p(x)) = 0$ if and only if $\mathit{fract}(\nu'_p(x)) = 0$, and
3. for each pair $x, y \in \pi^{-1}(p)$ such that $\nu_p(x) \leq C_x$ and $\nu_p(y) \leq C_y$, we have $\mathit{fract}(\nu_p(x)) \leq \mathit{fract}(\nu_p(y))$ if and only if $\mathit{fract}(\nu'_p(x)) \leq \mathit{fract}(\nu'_p(y))$.

From the result on timed automata [5], each $\sim_p$ is an equivalence relation and also a *time-abstract bisimulation*, i.e, if $\nu_p \sim_p \nu'_p$, then for all $t \in \mathbb{R}_{>0}$, there exists $t' \in \mathbb{R}_{>0}$, s.t., $\nu_p + t \sim \nu'_p + t'$.

Now, we say that two clock valuations ν and ν' over $\mathcal{Z}$ are *equivalent*, denoted $\nu \sim \nu'$ if they are equivalent when restricted to each process, i.e, $\nu_p \sim_p \nu'_p$ for all $p \in \mathit{Proc}$. An equivalence class of a clock valuation is called a *clock region* (of $\mathcal{B}$). For a valuation ν, $[\nu]$ denotes the clock region that contains ν. The set of clock regions of $\mathcal{B}$ is denoted by $\mathit{Regions}(\mathcal{B})$. The number of clock regions is finite: for instance, from [32] we have the bound $|\mathit{Regions}(\mathcal{B})| \leq ((2\,C + 2)^{|\mathcal{Z}|} \cdot |\mathcal{Z}|!)$, where C is the largest constant that a clock is compared with in $\mathcal{B}$.

Clearly, equivalent valuations satisfy the same guards and invariants: if $\nu \sim \nu'$ then $\nu \models \varphi$ if and only if $\nu' \models \varphi$ for all $\varphi \in \mathrm{Form}(\mathcal{Z})$. Moreover,

Lemma 7.6 (Time-abstract bisimulation)**.** *If $\nu \sim \nu'$, then for all $t \in \mathbb{R}_{>0}^{\mathit{Proc}}$, there exists $t' \in \mathbb{R}_{>0}^{\mathit{Proc}}$ such that $\nu + t \sim \nu' + t'$.*

Proof. Given $\nu, \nu' : \mathcal{Z} \to \mathbb{R}_{\geq 0}$ and $t = (t_p)_{p \in \mathit{Proc}}$ then $\nu \sim \nu'$ implies that $\nu_p \sim_p \nu'_p$ for each $p \in \mathit{Proc}$. Then, since each $\sim_p$ is a time-abstract bisimulation, for each $p \in \mathit{Proc}$, there exists $t'_p \in \mathbb{R}_{>0}$ such that $\nu_p + t_p \sim \nu'_p + t'_p$. Thus, defining $t' = (t'_p)_{p \in \mathit{Proc}}$ we obtain $\nu + t \sim \nu' + t'$. □

This equivalence can be naturally extended to states of $TS(\mathcal{B})$ by $(s, \nu) \sim (s', \nu')$ if $s = s'$ and $\nu \sim \nu'$. In order to show that this defines a bisimulation on $TS(\mathcal{B})$ (Proposition 7.7), we first introduce the successor relation on regions.

Let γ and γ' be two clock regions. We say that γ' is *accessible* from γ, written $\gamma \preceq \gamma'$, if either $\gamma' = \gamma$ or there are $\nu \in \gamma$, $\nu' \in \gamma'$, $t \in \mathbb{R}_{>0}^{\mathit{Proc}}$ such that $\nu' = \nu + t$. Note that $\preceq$ is a partial-order relation. The *successor* relation, written $\gamma \prec \gamma'$, is as usual defined by $\gamma \preceq \gamma'$, $\gamma \neq \gamma'$ and $\gamma'' = \gamma$ or $\gamma'' = \gamma'$ for all clock regions γ'' with $\gamma \preceq \gamma'' \preceq \gamma'$.

Example 38. The accessible-regions relation is illustrated in Figure 7.6. Suppose we deal with two processes, one owning clocks x_1 and x_2, the other owning a single clock y. Suppose furthermore that, in the icTA at hand, all clocks are compared to the constant 2. Consider the prisms $\gamma_0, \gamma_1, \gamma_2, \gamma'_1, \gamma'_2$, each representing a non-border clock region, which are given by the following clock constraints:

$$\gamma_0 = (0 < x_2 < x_1 < 1) \wedge (0 < y < 1)$$

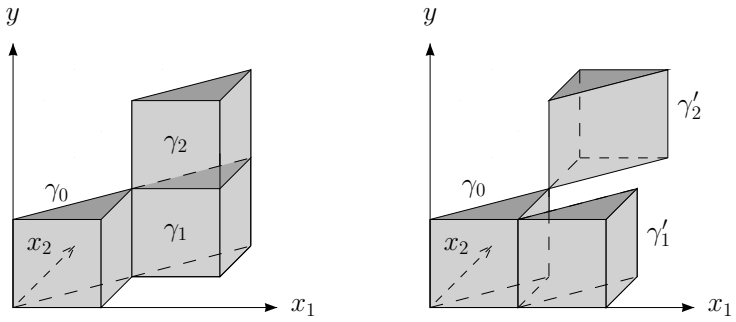


Figure 7.6: Accessible and non-accessible regions

$$\gamma_1 = (1 < x_2 < x_1 < 2) \wedge (0 < y < 1)$$

$$\gamma_2 = (1 < x_2 < x_1 < 2) \wedge (1 < y < 2)$$

$$\gamma_1' = (0 < x_2 < x_1 - 1 < 1) \wedge (0 < y < 1)$$

$$\gamma_2' = (1 < x_1 < x_2 < 2) \wedge (1 < y < 2)$$

We have $\gamma_0 \preceq \gamma_1 \preceq \gamma_2$. However, $\gamma_0 \not\preceq \gamma_1'$ and $\gamma_0 \not\preceq \gamma_2'$.

Proposition 7.7 (Bisimulation)**.** *If* $(s,\nu) \sim (s,\hat{\nu})$ *and* $(s,\nu) \xrightarrow{a} (s',\nu')$ *then* $(s,\hat{\nu}) \xrightarrow{a} (s',\hat{\nu}')$ *for some* $\hat{\nu}' \sim \nu'$*.*

Proof. Assume that $(s,\nu) \xrightarrow{a} (s',\nu')$. Let $t \in \mathbb{R}_{\geq 0}$, $\tau \in \mathit{Rates}$, $\varphi \in \mathrm{Form}(\mathcal{Z})$ and $R \subseteq \mathcal{Z}$ such that (7.6–7.10) hold. Consider the successive regions $\gamma_0 \prec \gamma_1 \prec \cdots \prec \gamma_n$ visited along $\nu + \tau[0 \ldots t]$: there is $0 = t_0 < t_1 < \cdots < t_n = t$ such that for $0 \leq i \leq n$ we have $\gamma_i = [\nu_i]$ with $\nu_i = \nu + \tau(t_i)$; and moreover, for any $1 \leq i \leq n$ and all $t_{i-1} < t' < t_i$ we have $\nu + \tau(t') \in \gamma_{i-1} \cup \gamma_i$.

Assume now in addition that $(s,\nu) \sim (s,\hat{\nu})$. We construct $\hat{\tau}$ such that for each $0 \leq i \leq n$ we have $P(i) : \hat{\nu}_i = \hat{\nu} + \hat{\tau}(t_i) \sim \nu_i$. We start with $\hat{\tau}(0) = 0$ so that $P(0)$ holds. Let now $1 \leq i \leq n$ and assume we have constructed $\hat{\tau}$ up to t_{i-1} with $P(i-1)$. Using Lemma 7.6 we find $\hat{t} \in \mathbb{R}_{>0}^{\mathit{Proc}}$ such that $\hat{\nu}_{i-1} + \hat{t} \sim \nu_{i-1} + \tau(t_i) - \tau(t_{i-1}) = \nu_i$. Then, define $\hat{\tau}$ on the interval $[t_{i-1}, t_i]$ using a linear interpolation so that $\hat{\tau}(t_i) = \hat{\tau}(t_{i-1}) + \hat{t}$. We obtain $\hat{\nu}_{i-1} + \hat{t} = \hat{\nu} + \hat{\tau}(t_i) = \hat{\nu}_i$ and $P(i)$ also holds. Finally, for $t' \geq t_n = t$, we let $\hat{\tau}(t') = \hat{\tau}(t_n) + \mathrm{id}(t' - t_n)$.

For any $1 \leq i \leq n$ and all $t_{i-1} < t' < t_i$ we have

$$\gamma_{i-1} = [\hat{\nu}_{i-1}] \preceq [\hat{\nu} + \hat{\tau}(t')] \preceq [\hat{\nu}_i] = \gamma_i$$

and since $\gamma_{i-1} \prec \gamma_i$ we obtain $\hat{\nu} + \hat{\tau}(t') \in \gamma_{i-1} \cup \gamma_i$. Therefore, $\hat{\nu} + \hat{\tau}(t') \models I(s)$ for all $t' \in [0,t]$ and $\hat{\nu}_n = \hat{\nu} + \hat{\tau}(t) \models \varphi$. We let $\hat{\nu}' = \hat{\nu}_n[R \to 0] \sim \nu_n[R \to 0] = \nu'$. We have $\hat{\nu}' \models I(s')$ and we deduce that $(s,\hat{\nu}) \xrightarrow{a} (s',\hat{\nu}')$ in $TS(\mathcal{B})$. □

To obtain the main result of this section, it remains to consider the finite quotient $TS(\mathcal{B})/\!\sim\; = (S'', \Sigma, \delta'', \iota'', F'')$. A state in S'' is an equivalence class $[(s,\nu)]$ and we have a transition $[(s,\nu)] \xrightarrow{a} [(s',\nu')] \in \delta''$ whenever $(s,\nu) \xrightarrow{a} (s',\nu')$

is a transition of $TS(\mathcal{B})$. The initial state is indeed $\iota'' = [(\iota, \nu_0)]$ where (ι, ν_0) is the initial state of $TS(\mathcal{B})$. Moreover, a state $[(s, \nu)]$ is final if and only if $s \in F$. Since the bisimulation equivalence relation $\sim$ on $TS(\mathcal{B})$ preserves final states, we obtain by standard (and easy) arguments:

Corollary 7.8. $\mathcal{L}(TS(\mathcal{B})/\!\sim) = \mathcal{L}(TS(\mathcal{B}))$.

The finite quotient $TS(\mathcal{B})/\!\sim$ is not exactly what is usually called the *region automaton* in the classical theory of timed automata. The main difference is that in the region automaton, transitions are decomposed into time-elapse ε-transitions from a region to a successor region, and discrete transitions with no time-elapse. The other minor difference is that we use as set of states $S' = S \times \mathit{Regions}(\mathcal{B})$ which is indeed isomorphic to S''. In particular, $\iota' = (\iota, [\nu_0])$ and $F' = F \times \mathit{Regions}(\mathcal{B})$. The region automaton associated with $\mathcal{B}$ is therefore $\mathcal{R}_\mathcal{B} = (S', \Sigma, \delta', \iota', F')$ where, for $a \in \Sigma_\varepsilon$ and $s, s' \in S$ and $\gamma, \gamma' \in \mathit{Regions}(\mathcal{B})$, δ' contains $(s, \gamma) \xrightarrow{a} (s', \gamma')$ if

- $a = \varepsilon$, $s = s'$, $\gamma \prec \gamma'$, and $\nu' \models \mathit{Inv}(s)$ for some $\nu' \in \gamma'$
  (we then call $(s, \gamma) \xrightarrow{\varepsilon} (s, \gamma')$ a *time-elapse transition*), or
- there are $\nu \in \gamma$ and $(s, \varphi, a, R, s') \in \delta$ such that $\nu \models \varphi \wedge \mathit{Inv}(s)$, $\nu[R \to 0] \models \mathit{Inv}(s')$, and $\nu[R \to 0] \in \gamma'$ (we then call $(s, \gamma) \xrightarrow{a} (s', \gamma')$ a *discrete transition*).

A part of the region automaton for the icTA from Figure 7.3 is shown in Figure 7.15.

It is easy to see that a sequence of time-elapse transitions followed by a discrete transition of $\mathcal{R}_\mathcal{B}$ is a transition of $TS(\mathcal{B})/\!\sim$. Conversely, any transition of $TS(\mathcal{B})/\!\sim$ can be decomposed into a sequence of time-elapse transitions followed by a discrete transition of $\mathcal{R}_\mathcal{B}$. Therefore,

Theorem 7.9. *Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, \mathit{Inv}, \iota, F, \pi)$ be an* icTA *and let C be the largest constant a clock is compared with in $\mathcal{B}$. Then, the number of states of $TS(\mathcal{B})/\!\sim$ and of $\mathcal{R}_\mathcal{B}$ is bounded by $|S| \cdot (2\,C + 2)^{|\mathcal{Z}|} \cdot |\mathcal{Z}|!$ and we have*

$$\mathcal{L}(\mathcal{R}_\mathcal{B}) = \mathcal{L}(TS(\mathcal{B})/\!\sim) = \mathcal{L}(TS(\mathcal{B})) = \mathcal{L}_\exists(\mathcal{B})$$

which is therefore a regular word language.

Now, if we are given a negative specification as a regular set *Bad*. Then since $\mathcal{L}_\exists(\mathcal{B})$ is a regular word language, so is $\mathcal{L}_\exists(\mathcal{B}) \cap \mathit{Bad}$ and hence we can check emptiness and so on. Thus, we solved the verification problem stated at the beginning of this section:

Theorem 7.10. *Model checking* icTAs *wrt. regular negative specifications is decidable.*

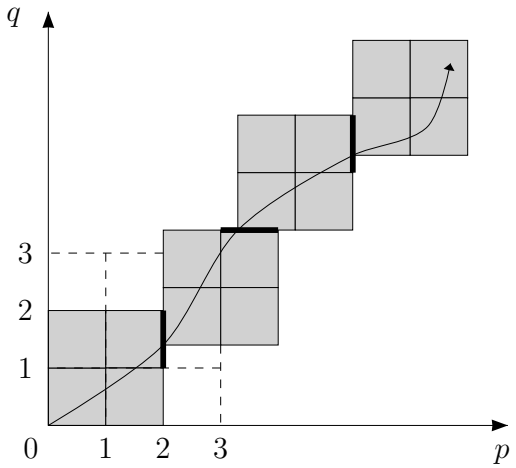


Figure 7.7: $dir(\tau) = 010\ldots$

7.3 The universal semantics

While the existential semantics allows us to verify negative specifications, the universal semantics is natural when we want to check if our system has some good behavior. By good we mean a behavior that is robust against clock variations. Unfortunately, we show in this section that emptiness and universality are undecidable for the universal semantics. This is shown for icTAs first and then will be extended to DTAs. Therefore, it is undecidable if, for a positive specification *Good* containing the behaviors that a system *must* exhibit and a DTA $\mathcal{D}$, we have $Good \subseteq \mathcal{L}_\forall(\mathcal{D})$.

Theorem 7.11. *The following problem is undecidable if* $|Proc| \geq 2$*: Given an* icTA $\mathcal{B}$ *over Proc, does* $\mathcal{L}_\forall(\mathcal{B}) \neq \emptyset$ *hold?*

Proof. The proof is by reduction from Post's correspondence problem (PCP). An instance *Inst* of the PCP consists of an alphabet A and two morphisms f and g from A^+ to $\{0,1\}^+$. A solution of *Inst* is a word $w \in A^+$ such that $f(w) = g(w)$.

Suppose $Proc = \{p, q\}$ and let $\tau \in Rates$. One may associate with τ two sequences $t\text{-}dir(\tau) = t_1 t_2 \ldots \in (\mathbb{R}_{\geq 0})^\omega$ of time instances and $dir(\tau) = d_1 d_2 \ldots \in \{0,1,2\}^\omega$ of *directions* as follows: for $i \geq 1$, we let first (assuming $t_0 = 0$) $t_i = \min\{t > t_{i-1} \mid \tau_r(t) - \tau_r(t_{i-1}) = 2 \text{ for some } r \in Proc\}$. With this, we set

$$d_i = \begin{cases} 0 & \text{if } \tau_p(t_i) - \tau_p(t_{i-1}) = 2 \text{ and } 1 < \tau_q(t_i) - \tau_q(t_{i-1}) < 2 \\ 1 & \text{if } \tau_q(t_i) - \tau_q(t_{i-1}) = 2 \text{ and } 1 < \tau_p(t_i) - \tau_p(t_{i-1}) < 2 \\ 2 & \text{otherwise} \end{cases}$$

The construction of $dir(\tau)$ is illustrated in Figure 7.7. The idea is to allow the shape of the relative time-rate function (from τ) to encode a word in $\{0,1,2\}^\omega$. We do this using 2×2-square regions, each consisting of 4 sub-squares as shown. If the rate function leaves this region by the upper boundary or right boundary of the right-upper sub-square, then we write 1 or 0, respectively. If it leaves by any other boundary or by end-points of any sub-square, then we write 2. A new square

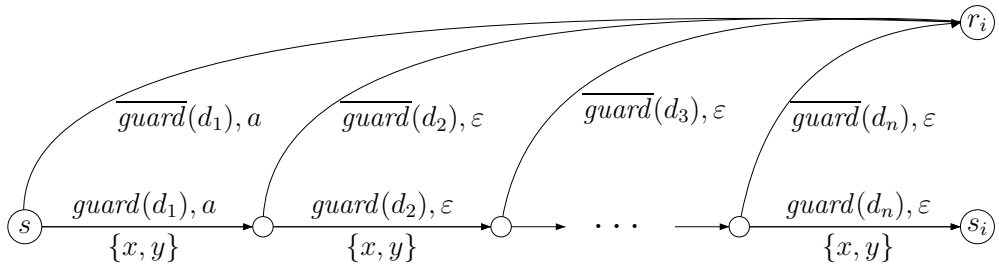


Figure 7.8: Transition macro

region is started at the point where the rate function left the old one. Thus, the direction sequences partition the space of time rates.

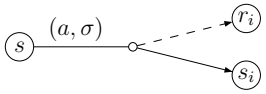
Roughly speaking, a word is accepted universally by an icTA iff it is accepted for all directions. Our trick will be to define an icTA, where the invariants, guards and resets are used in a restricted way, so that, the PCP instance has a solution w iff the word wb is accepted by the icTA for all directions. Thus, if there is no solution to the PCP, there will be some direction sequence (respectively, local time rates) for which the icTA does not accept.

Let an instance *Inst* of the PCP be given by an alphabet $A = \{a_1, \ldots, a_k\}$ with $k \geq 1$ and two corresponding morphisms f and g. We will construct an icTA $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ over the set of processes $Proc = \{p, q\}$ and $\Sigma = \{a_1, \ldots, a_k, b\}$ such that $\mathcal{L}_\forall(\mathcal{B}) = \{wb \mid w \in A^+ \text{ and } f(w) = g(w)\}$. First, let $\mathcal{Z} = \{x, y\}$ with $\pi(x) = p$ and $\pi(y) = q$. For $d \in \{0, 1, 2\}$, we set

$$guard(d) = \begin{cases} x = 2 \ \wedge \ 1 < y < 2 & \text{if } d = 0 \\ y = 2 \ \wedge \ 1 < x < 2 & \text{if } d = 1 \\ ((x \leq 1 \ \vee \ x = 2) \ \wedge y = 2) \ \vee \ (y \leq 1 \ \wedge \ x = 2) & \text{if } d = 2 \end{cases}$$

Moreover, let $\overline{guard}(d) = \bigvee_{d' \in \{0,1,2\} \setminus \{d\}} guard(d')$.

The final encoding of the given PCP instance in terms of the icTA is given by Figure 7.9. Hereby, given $a \in A$, $\sigma = d_1 \ldots d_n \in \{0, 1, 2\}^+$ (with $d_j \in \{0, 1, 2\}$ for any $j \in \{1, \ldots, n\}$) and $i \in \{1, 2\}$, a transition of the form



will actually stand for the sequence of transitions that is depicted in Figure 7.8, say, with intermediate states $s_{(i,a,\sigma,1)}, \ldots, s_{(i,a,\sigma,n-1)}$.

Example 39. Consider the PCP instance *Inst* given by $A = \{a_1, a_2\}$, $f(a_1) = 101$, $g(a_1) = 1$, $f(a_2) = 1$, $g(a_2) = 01110$ with the obvious solution $w = a_1a_2a_1$. One can check that $a_1a_2a_1b \in \mathcal{L}_\forall(\mathcal{B})$. This is illustrated in Figure 7.10. In the tree depicted, any path corresponds to a finite prefix (of length $|f(w)| + 1$) of some sequence of directions. The edges are labeled by this sequence, where a left-edge

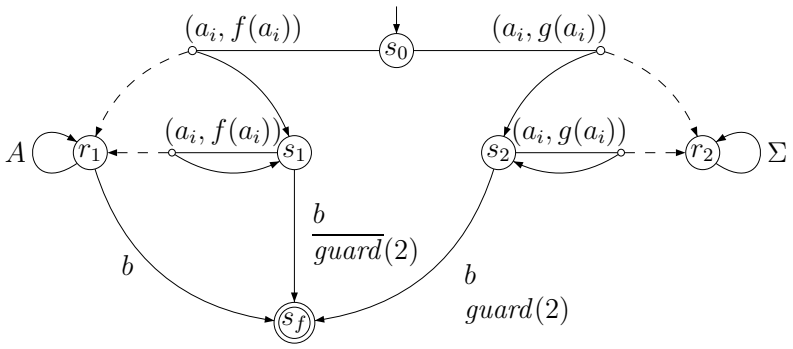


Figure 7.9: Encoding of PCP: icTA $\mathcal{B}$

is 0, downward is 2 and right-edge is 1. Thus, intuitively, a word wb is in the universal language iff all paths of the tree correspond to accepting runs in $\mathcal{B}$. Now, lets verify that the word wb is accepted by $\mathcal{B}$. If clock rate τ is such that $dir(\tau) \in f(w) \cdot d \cdot \{0,1,2\}^\omega$ with $d \in \{0,1\}$, then the accepting run of $\mathcal{B}$ is the path shown in the left figure, which assigns states s_1 to nodes of the tree and finishes at s_f. If $d = 2$, then the accepting run of $\mathcal{B}$ is the path in the figure on right, which assigns states s_2 appropriately, crucially using the fact that $f(w) = g(w)$, and finally ends at s_f. If the clock rate τ has $dir(\tau)$ different from the above cases, it is easy to see that there is an accepting run in which $\mathcal{B}$ reaches state s_f by passing through state r_1.

Let us show that our reduction is indeed correct. In the following, let $\leq$ denote the usual prefix relation on words. We begin by observing that if a τ-run starting from s_0 on $\mathcal{B}$ satisfies the guards given by $guard(d)$ (and thus avoids states r_1 and r_2), then the time stamps of the run are exactly the ones given by $dir(\tau)$.

Claim 7.12. *Let $\tau \in Rates$ and let $t\text{-}dir(\tau) = t_1 t_2 \ldots \in (\mathbb{R}_{\geq 0})^\omega$ and $dir(\tau) = d_1 d_2 \ldots \in \{0,1,2\}^\omega$ be the associated sequences. In addition, we set $t_0 = 0$. Consider a τ-run*

$$(s_0, \nu_0) \xrightarrow{a_1, t'_1} (u_1, \nu_1) \ldots (u_{n-1}, \nu_{n-1}) \xrightarrow{a_n, t'_n} (u_n, \nu_n)$$

of $\mathcal{B}$ with $a_i \in \Sigma_\varepsilon$ and where $u_{n-1} \notin \{r_1, r_2, s_f\}$. Assume moreover $d'_1, \ldots, d'_n$ are the (unique) elements from $\{0,1,2\}$ such that $\nu_{i-1} + \tau(t'_i) - \tau(t'_{i-1}) \models guard(d'_i)$ for all $i \in \{1, \ldots, n\}$. Then, $t'_i = t_i$ and $d'_i = d_i$ for all $1 \leq i \leq n$, and $\nu_i = \nu_0$ for all $1 \leq i < n$.

Proof. We proceed by induction. Assume $t'_{i-1} = t_{i-1}$ and $\nu_{i-1}(x) = \nu_{i-1}(y) = 0$ for some $1 \leq i \leq n$ (note that this is the case for $i = 1$). Since we have a τ-run, $\tau(t'_i) - \tau(t_{i-1}) \models guard(d'_i)$. Here we consider the former expression as a valuation by considering the p-component of the pair as the clock value of x

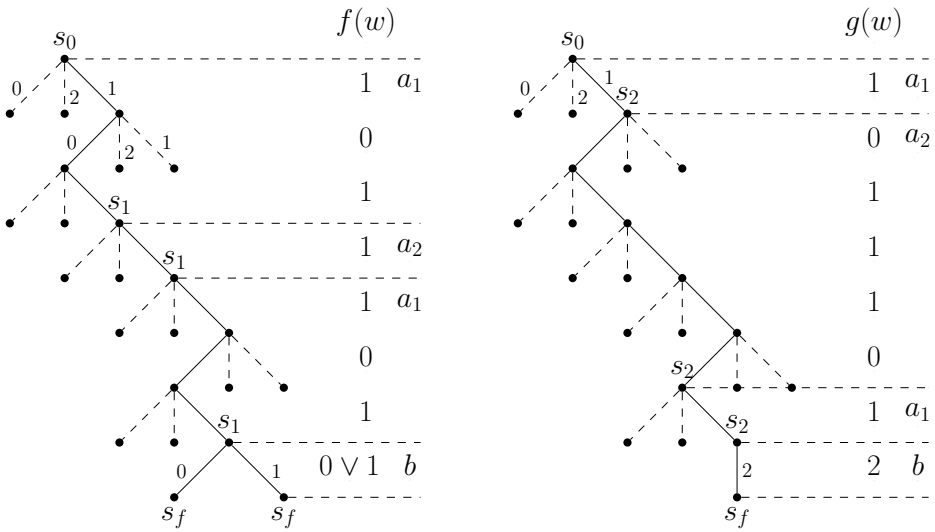


Figure 7.10: The tree generated by $w = a_1a_2a_1b$ with respect to f and g.

and the q-value as that of y. Now assume $d'_i = 0$ (the cases $d'_i \in \{1, 2\}$ are analogous). Then, $\tau_p(t'_i) - \tau_p(t_{i-1}) = 2$ and $1 < \tau_q(t'_i) - \tau_q(t_{i-1}) < 2$. Hence, $t'_i = \min\{t > t_{i-1} \mid \tau_r(t) - \tau_r(t_{i-1}) = 2 \text{ for some } r \in \mathit{Proc}\}$. We deduce $t'_i = t_i$ and $d'_i = d_i$. Moreover, $\nu_i(x) = \nu_i(y) = 0$ if $i < n$. □

Claim 7.13. *For $\tau \in \mathit{Rates}$ and $w \in A^+$, the following hold:*

(1) $f(w) \leq \mathit{dir}(\tau)$ *iff* $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1$

(2) $g(w) \leq \mathit{dir}(\tau)$ *iff* $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2$

(3) $f(w) \not\leq \mathit{dir}(\tau)$ *iff* $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1$

Proof. First note that (1) and (2) can be shown along the same lines. Moreover, their "if"- directions follow from Claim 7.12.

Let $\tau \in \mathit{Rates}$ and let $t\text{-}\mathit{dir}(\tau) = t_1t_2 \ldots \in (\mathbb{R}_{\geq 0})^\omega$ and $\mathit{dir}(\tau) = d_1d_2 \ldots \in \{0, 1, 2\}^\omega$ be the associated sequences. We also set $t_0 = 0$. Suppose $w = a_1 \ldots a_n$ where $a_i \in A$ for all $i \in \{1, \ldots, n\}$. For $i \in \{1, \ldots, n\}$, let moreover ℓ_i denote the length of $f(a_1 \ldots a_i)$. Finally, ν is the valuation with $\nu(x) = \nu(y) = 0$.

(1) Suppose $f(w) \leq \mathit{dir}(\tau)$. We have $f(w) = d_1 \ldots d_{\ell_n}$. Let us check that

$$\begin{array}{llllll}
(s_0, \nu) & \xrightarrow{(a_1, t_1)} & (s_{(1, a_1, f(a_1)), 1)}, \nu) & \xrightarrow{(\varepsilon, t_2)} & \ldots & \xrightarrow{(\varepsilon, t_{\ell_1})} (s_1, \nu) \\
& \xrightarrow{(a_2, t_{\ell_1+1})} & (s_{(1, a_2, f(a_2)), 1)}, \nu) & \xrightarrow{(\varepsilon, t_{\ell_1+2})} & \ldots & \xrightarrow{(\varepsilon, t_{\ell_2})} (s_1, \nu) \\
& & \vdots & & & \\
& \xrightarrow{(a_2, t_{\ell_{n-1}+1})} & (s_{(1, a_n, f(a_n)), 1)}, \nu) & \xrightarrow{(\varepsilon, t_{\ell_{n-1}+2})} & \ldots & \xrightarrow{(\varepsilon, t_{\ell_n})} (s_1, \nu)
\end{array}$$

is a τ-run of $\mathcal{B}$, which implies $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1$. We need to show that, for all $i \in \{1, \ldots, \ell_n\}$, $\tau(t_i) - \tau(t_{i-1}) \models \mathit{guard}(d_i)$. So let $i \in \{1, \ldots, \ell_n\}$. Assuming

$d_i = 0$, we indeed have $\tau(t_i) - \tau(t_{i-1}) \models x = 2 \wedge 1 < y < 2$, whereas $d_i = 1$ implies $\tau(t_i) - \tau(t_{i-1}) \models y = 2 \wedge 1 < x < 2$.

(3) Let us assume $f(w) \not\leq dir(\tau)$. There is $j \in \{1 \dots, \ell_n\}$ such that we have $f(w) = d_1 \dots d_{j-1} d'_j \dots d'_{\ell_n}$ for some $d'_j, \dots, d'_{\ell_n} \in \{0, 1\}$ with $d'_j \neq d_j$. We construct a τ-run that coincides with the run that we constructed above until the $(j-1)$-th transition. Now, $\tau(t_j) - \tau(t_{j-1}) \models guard(d_j)$ and since $d'_j \neq d_j$, the j-th transition leads to state r_1. Once in r_1, we stay in r_1 under any timing. Thus, $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1$. Conversely, assume that $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1$ and consider a τ-run

$$(s_0, \nu_0) \xrightarrow{a'_1, t'_1} (u_1, \nu_1) \xrightarrow{(a'_2, t'_2)} \cdots \xrightarrow{(a'_{j-1}, t'_{j-1})} (u_{j-1}, \nu_{j-1}) \xrightarrow{a'_j, t'_j} (r_1, \nu')$$

with $a'_1 \cdots a'_j \leq w$ and $u_{j-1} \neq r_1$. Let moreover $d'_1, \dots, d'_j$ be the (unique) elements from $\{0, 1\}$ such that $\nu_{i-1} + \tau(t'_i) - \tau(t'_{i-1}) \models guard(d'_i)$ for all $1 \leq i \leq j$. By Claim 7.12, we deduce that $t_i = t'_i$ and $d'_i = d_i$ for all $1 \leq i \leq j$. Since the last transition reaches state r_1 from $u_{j-1} \neq r_1$, we deduce that $d_1 \cdots d_{j-1} \leq f(w)$ but d_j differs from the j-th letter of $f(w)$. Therefore, $f(w) \not\leq dir(\tau)$. □

With Claim 7.13, we can now show both directions of the correctness of the construction of $\mathcal{B}$: $\mathcal{L}_\forall(\mathcal{B}) = \{wb \mid w \in A^+ \text{ and } f(w) = g(w)\}$.

Let $w \in A^+$ with $f(w) = g(w)$ and let $\tau \in Rates$. We distinguish three cases. If $dir(\tau) \in f(w) \cdot \{0, 1\} \cdot \{0, 1, 2\}^\omega$, then $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1 \xrightarrow{b} s_f$ by Claim 7.13 (1). If $dir(\tau) \in f(w) \cdot 2 \cdot \{0, 1, 2\}^\omega$, then $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2 \xrightarrow{b} s_f$ by Claim 7.13 (2), since $g(w) = f(w)$. If $f(w) \not\leq dir(\tau)$, then $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} r_1 \xrightarrow{b} s_f$ by Claim 7.13 (3). Hence, $wb \in \mathcal{L}_\forall(\mathcal{B})$.

Conversely, let $w \in A^+$ and suppose $wb \in \mathcal{L}_\forall(\mathcal{B})$. Pick $\tau \in Rates$ such that $dir(\tau) \in f(w) \cdot 2 \cdot \{0, 1, 2\}^\omega$. As $f(w) \leq dir(\tau)$, we have $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1$ and $(\mathcal{B}, \tau) : s_0 \not\xrightarrow{w} r_1$ by Claim 7.13 (1,3), and since $f(w) \cdot 2 \leq dir(\tau)$ we deduce that $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_1 \not\xrightarrow{b}$. Thus, we must have $(\mathcal{B}, \tau) : s_0 \xrightarrow{w} s_2 \xrightarrow{b} s_f$. Hence, $g(w) \cdot 2 \leq dir(\tau)$ by Claim 7.13 (2). As $f(w), g(w) \in \{0, 1\}^*$ and we have both $f(w) \cdot 2 \leq dir(\tau)$ and $g(w) \cdot 2 \leq dir(\tau)$, we deduce that $f(w) = g(w)$. □

Thus we have shown that it is undecidable to check the emptiness of universal semantics. We also show that, the universality problem for this case continues to be undecidable.

Theorem 7.14. *Suppose that $|Proc| \geq 2$. For* icTA *$\mathcal{B}$ over Proc, it is undecidable to know if $\mathcal{L}_\forall(\mathcal{B}) = \Sigma^*$ (where Σ is the set of actions of $\mathcal{B}$).*

Proof. As before, the reduction is from the PCP problem. The construction of the corresponding automaton is obtained by a slight modification of the automaton in the previous proof as we shall see below. We consider, as before, an instance of

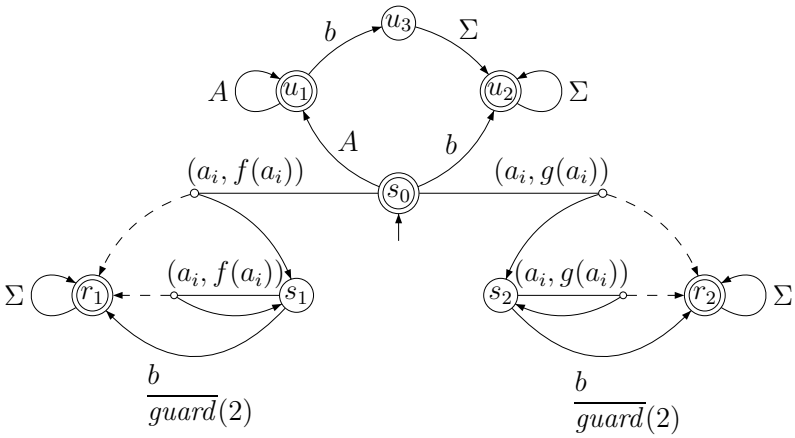


Figure 7.11: Encoding of PCP: icTA $\widetilde{\mathcal{B}}$

the PCP. We will then construct an icTA $\widetilde{\mathcal{B}}$ over $Proc = \{p, q\}$ and $\Sigma = A \cup \{b\}$, where $A = \{a_1, \ldots, a_k\}$, such that

$$\mathcal{L}_\forall(\widetilde{\mathcal{B}}) = \Sigma^* \setminus \{wb \mid w \in A^+ \text{ and } f(w) = g(w)\}.$$

Thus, the PCP instance has a solution iff $\mathcal{L}_\forall(\widetilde{\mathcal{B}}) \neq \Sigma^*$.

Define icTA $\widetilde{\mathcal{B}}$ as in Figure 7.11 where the transition macros are as defined in Figure 7.8. This automaton is almost the same as $\mathcal{B}$ in Figure 7.9, except for two differences. One is that we have switched the final states and b-transitions leading to them. Further, we have a 3-state gadget on the top of the previous automaton which does not use any clocks. In fact, this gadget is just the automaton which accepts the language $\Sigma^* \setminus A^+b$. Thus, if we have a word which is not in A^+b, it gets nondeterministically accepted by this gadget. If the word is in A^+b then it can only be accepted in states r_1 or r_2.

Claim 7.15. *For $\tau \in Rates$, $w \in A^+$,*

(a) $f(w) \cdot 2 \not\leq dir(\tau)$ iff $(\widetilde{\mathcal{B}}, \tau) : s_0 \xrightarrow{wb} r_1$

(b) $g(w) \cdot 2 \not\leq dir(\tau)$ iff $(\widetilde{\mathcal{B}}, \tau) : s_0 \xrightarrow{wb} r_2$

Proof. The proof of both these statements follows that from Claim 7.13 which itself depends on Claim 7.12.. But, since we have only fiddled with the final states/transitions, these claims still hold. □

We resume with the proof of Theorem 7.14. First, suppose the PCP has a solution, say $w \in A^+$, then consider the word wb and choose τ such that $dir(\tau) \in f(w) \cdot 2 \cdot \{0, 1, 2\}^\omega$. Then, by Claim 7.15, we have $(\widetilde{\mathcal{B}}, \tau) : s_0 \not\xrightarrow{wb} r_1$. Now, since w is assumed to be a solution, we have $f(w) = g(w)$ and $dir(\tau) \in g(w) \cdot 2 \cdot \{0, 1, 2\}^\omega$

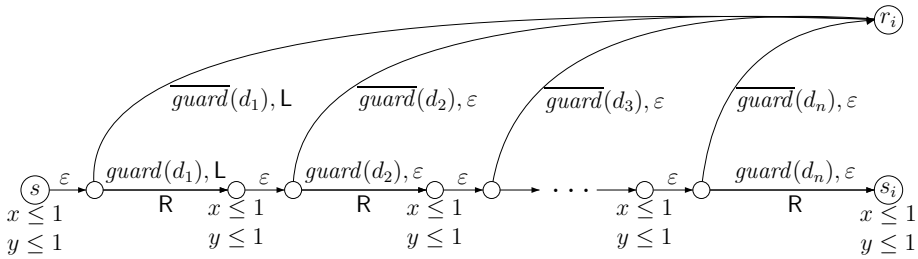


Figure 7.12: Transition macro for the distributed setting

and so again from Claim 7.15 we have $(\widetilde{\mathcal{B}}, \tau) : s_0 \xrightarrow{wb} r_2$. Further, on reading wb, $\widetilde{\mathcal{B}}$ can only (nondeterministically) reach the reject state u_3, not the accepting states u_1 or u_2 or s_0. Therefore, $wb \notin \mathcal{L}(\widetilde{\mathcal{B}}, \tau)$ and so $\mathcal{L}_\forall(\widetilde{\mathcal{B}}) \neq \Sigma^*$.

Conversely, suppose $\mathcal{L}_\forall(\widetilde{\mathcal{B}}) \neq \Sigma^*$. Let $w' \in \Sigma^*$ such that $w' \notin \mathcal{L}_\forall(\widetilde{\mathcal{B}})$. Necessarily, $w' = wb$ with $w \in A^+$, since otherwise it would be accepted by the gadget for all $\tau \in \mathit{Rates}$. Moreover, there exists τ such that $w' \notin \mathcal{L}(\widetilde{\mathcal{B}}, \tau)$, i.e., after reading $w' = wb$, $\widetilde{\mathcal{B}}$ does not reach the accepting states r_1 or r_2. By Claim 7.15, we can now conclude that $f(w) \cdot 2 \leq \mathit{dir}(\tau)$ and $g(w) \cdot 2 \leq \mathit{dir}(\tau)$, which finally implies that $f(w) = g(w)$ and thus the PCP has a solution, namely w. □

In fact, the above proof also demonstrates that checking if the existential and universal semantics coincide is undecidable.

Corollary 7.16. *Suppose that* $|\mathit{Proc}| \geq 2$. *For* icTA $\mathcal{B}$ *over* Proc, *it is undecidable to check if* $\mathcal{L}_\forall(\mathcal{B}) = \mathcal{L}_\exists(\mathcal{B})$.

Proof. Consider the icTA $\widetilde{\mathcal{B}}$ in Figure 7.11 constructed for the above proof. Then $\mathcal{L}_\exists(\widetilde{\mathcal{B}}) = \Sigma^*$ and thus checking if $\mathcal{L}_\forall(\widetilde{\mathcal{B}}) = \mathcal{L}_\exists(\widetilde{\mathcal{B}})$ is the same as checking if $\mathcal{L}_\forall(\widetilde{\mathcal{B}}) = \Sigma^*$. But this is undecidable by Theorem 7.14. □

These results can be strengthened and extended to the distributed setting as follows:

Theorem 7.17. *Suppose* $|\mathit{Proc}| \geq 2$. *For DTAs* $\mathcal{D}$ *over* Proc, *the emptiness and universality of* $\mathcal{L}_\forall(\mathcal{D})$ *are undecidable.*

Proof. We fix $\mathit{Proc} = \{p, q\}$ and the clock distribution $\pi(x) = p$ and $\pi(y) = q$. Each process will be a copy of the automaton $\mathcal{B}$ that is depicted in Figure 7.9 for emptiness (respectively, $\widetilde{\mathcal{B}}$ in Figure 7.11 for universality), except for one difference: in the copy $\mathcal{B}_p$ for process p, the transition macro from Figure 7.8 is replaced with that from Figure 7.12 where L is the letter $a \in A$ and R is the singleton set $\{x\}$; and in the copy $\mathcal{B}_q$ for process q, we use the same new macro, but now we have $\mathsf{L} = \varepsilon$ and $\mathsf{R} = \{y\}$.

The main difficulty is to make sure that transitions with $\mathit{guard}(d_j)$ or $\overline{\mathit{guard}}(d_j)$ are taken simultaneously in the two copies $\mathcal{B}_p$ and $\mathcal{B}_q$. If this is the case, then clock

y is always reset synchronously with clock x and $\mathcal{B}_p$ faithfully simulates the icTA $\mathcal{B}$ (or $\widetilde{\mathcal{B}}$) with the slight difference induced by the additional ε-transitions from the states with invariant $x \leq 1 \wedge y \leq 1$. Therefore, the proof of Theorems 7.11 or 7.14 can be carried out similarly.

We explain now how we make sure that transitions with guards are taken simultaneously. We have splitted each state of the transition macro described in Figure 7.8 (except s_i and r_i) into two. The combination of the guards and the invariants ensure that both clocks have been reset simultaneously.

Let us examine this in more detail. Being in two identical copies of a state with an invariant, the ε-transitions might indeed be taken asynchronously by $\mathcal{B}_p$ and $\mathcal{B}_q$. However, the following transitions will be performed synchronously. Assume first that p follows a transition of the form $(s_p, \mathit{guard}(d), a, \{x\}, s'_p)$ before process q moves. As $\mathit{guard}(d)$, where $d \in \{0, 1\}$, is satisfied when p goes to s'_p, the value of both clocks exceeds 1. But as x is reset at the same time whereas y is not, the invariant associated with s'_p is violated, which is a contradiction. Thus, q has to take the corresponding transition, which is of the form $(s_q, \mathit{guard}(d), a, \{y\}, s'_q)$, simultaneously. This explains why we use 2×2-squares as in Figure 7.7 and corresponding guards. In the DTA $\mathcal{D}$, they allow us to check when one clock has been reset and the other has not. Now consider the case where p performs a transition of the form $(s_p, \overline{\mathit{guard}}(d), a, \emptyset, s'_p)$. When p executes its transition, at least one clock has reached the value 2. As this clock cannot be reset anymore, q is obliged to follow instantaneously the corresponding transition of the form $(s_q, \overline{\mathit{guard}}(d), a, \emptyset, s'_q)$, to reach a final state. □

Corollary 7.18. *Given a DTA $\mathcal{D}$ and a regular positive specifications Good, it is undecidable to check whether Good $\models \mathcal{L}_\forall(\mathcal{D})$.*

7.4 Playing with local time rates

In the previous sections, we considered the set of behaviours when the local-time rates are arbitrarily chosen (existential) or arbitrarily enforced (universal). It is then natural to ask what would happen if there are some restrictions on the way these rates are chosen or enforced. In this section, we consider some such questions. Broadly, we examine two types of restrictions on the local-time rates. In the first case, we try to bound the clock drifts between processes, while in the second case, we restrict to a natural sub-class of local-time rate functions.

7.4.1 Bounding the clock drifts

Our first attempt is to try to bound the way the clocks drift on different processes, thus curtailing the independence of the local-time rates. We consider two sub-cases

Figure 7.13: Bounding the clock drifts by ratio and difference

here, where we insist (1) the ratio or (2) the difference of local times in different processes is always bounded by a constant.

We might expect that such a strong restriction could lead to a decidability result for the universal semantics. However, it turns out that we can strengthen the undecidability proof in Section 7.3 to show that emptiness and universality of the universal semantics is already undecidable in this restricted case.

Let us formalize this. We will restrict to two processes, $Proc = \{p, q\}$. We note however that the following definitions (and results) can easily be generalized to more processes. For a rational number $k \geq 1$, we define $Rates_{\text{rat}}(k) = \{\tau = (\tau_p, \tau_q) \in Rates \mid \frac{1}{k} \leq \frac{\tau_p(t)}{\tau_q(t)} \leq k \text{ for all } t \in \mathbb{R}_{>0}\}$. This is the set of all rate-function tuples such that the ratio of the local times in the two processes are always bounded by fixed rationals. Thus, when we plot the local times in the two components of the rate-function tuple against each other, this function lies completely within the shaded region in Figure 7.13 (a).

Further, for a rational number $\ell \geq 0$, $Rates_{\text{diff}}(\ell) = \{\tau = (\tau_p, \tau_q) \in Rates \mid |\tau_p(t) - \tau_q(t)| \leq \ell \text{ for all } t \in \mathbb{R}_{\geq 0}\}$. These are the rate-function tuples for which the difference between the local times in the two processes are bounded by some constant. Thus again, any function in the shaded region in Figure 7.13 (b) describes such a bounded rate-function tuple.

Accordingly, for an icTA or a DTA $\mathcal{B}$, we define

- $\mathcal{L}_{\exists}^{\text{rat},k}(\mathcal{B}) = \bigcup_{\tau \in Rates_{\text{rat}}(k)} \mathcal{L}(\mathcal{B}, \tau)$, $\mathcal{L}_{\forall}^{\text{rat},k}(\mathcal{B}) = \bigcap_{\tau \in Rates_{\text{rat}}(k)} \mathcal{L}(\mathcal{B}, \tau)$,
- $\mathcal{L}_{\exists}^{\text{diff},\ell}(\mathcal{B}) = \bigcup_{\tau \in Rates_{\text{diff}}(\ell)} \mathcal{L}(\mathcal{B}, \tau)$, $\mathcal{L}_{\forall}^{\text{diff},\ell}(\mathcal{B}) = \bigcap_{\tau \in Rates_{\text{diff}}(\ell)} \mathcal{L}(\mathcal{B}, \tau)$.

Theorem 7.19. *For* icTAs *or DTAs* $\mathcal{B}$ *over* $Proc = \{p, q\}$,

1. *checking emptiness of* $\mathcal{L}_{\forall}^{\text{rat},1}(\mathcal{B}) = \mathcal{L}_{\forall}^{\text{diff},0}(\mathcal{B})$ *is decidable while checking universality is undecidable.*
2. *checking emptiness and universality of* $\mathcal{L}_{\forall}^{\text{rat},k}(\mathcal{B})$ *are undecidable for every rational* $k > 1$.

3. *checking emptiness and universality of* $\mathcal{L}_\forall^{\text{diff},\ell}(\mathcal{B})$ *are undecidable for every rational* $\ell > 0$.

Proof. For $k = 1$ or $\ell = 0$, the sets $Rates_{\text{rat}}(k)$ and $Rates_{\text{diff}}(\ell)$ consist of exactly the tuples in which time evolves at the same rate in both processes. Thus, the sets $\mathcal{L}_\forall^{\text{rat},1}(\mathcal{B})$ and $\mathcal{L}_\forall^{\text{diff},0}(\mathcal{B})$ are identical and correspond to the language of an ordinary timed automaton. Hence, checking emptiness is decidable, while checking universality is undecidable. This proves Part (2) of the theorem.

To prove the remaining parts of the theorem, we need the following lemma.

Lemma 7.20. *Let* $k > 1$, $\ell > 0$ *be some fixed rationals. For all* $\sigma \in \{0,1,2\}^*$, *there exists* $\tau \in Rates_{\text{rat}}(k) \cap Rates_{\text{diff}}(\ell)$ *such that* σ *is a prefix of* $dir(\tau)$.

Proof. Let $\sigma = d_1 d_2 \dots d_n \in \{0,1,2\}^*$ be of length n. We define τ (in terms of $n+1$ points) as follows: τ_p is the piecewise linear function with $\tau_p(2i) = x_i$ for $i \in \{0,\dots,n\}$ and $\tau_p(2n+t) = x_n + t$ for all $t \in \mathbb{R}_{\geq 0}$. Similarly, τ_q is defined as the piecewise linear function with $\tau_q(2i) = y_i$ for $i = \{0,\dots,n\}$ and $\tau_q(2n+t) = y_n + t$ for $t \in \mathbb{R}_{\geq 0}$. The points (x_i, y_i) are defined by $x_0 = y_0 = 0$ and, for $i \in \{1,\dots,n\}$, $x_i = 2i - \alpha|d_1 \dots d_i|_1$ and $y_i = 2i - \alpha|d_1 \dots d_i|_0$ ($|\sigma'|_d$ denoting the number of occurrences of d in σ'), where α is a rational parameter to be fixed.

With the above definition, we observe that, for all i, we have $|x_i - y_i| \leq i\alpha$, and, for $i > 0$, we have $1 - \frac{\alpha}{2} \leq \frac{x_i}{y_i} \leq \frac{1}{1-\alpha/2}$. Thus, by choosing $\alpha = \min\{\frac{1}{2}, \frac{\ell}{n}, 2(1-\frac{1}{k})\}$, we can check that $\tau \in Rates_{\text{rat}}(k) \cap Rates_{\text{diff}}(\ell)$.

Finally, we show that $dir(\tau) = \sigma \cdot 2^\omega$. This is done by induction. Assume that $d_1 \cdots d_{i-1} \leq dir(\tau)$ for some $1 \leq i \leq n$. If $d_i = 2$ then $x_i - x_{i-1} = 2 = y_i - y_{i-1}$ and we deduce that $d_1 \cdots d_{i-1} d_i \leq dir(\tau)$. If now $d_i = 0$ then $x_i - x_{i-1} = 2$ and $y_i - y_{i-1} = 2 - \alpha$. Since $0 < \alpha < 1$, we deduce that $d_1 \cdots d_{i-1} d_i \leq dir(\tau)$. The proof is similar when $d_i = 1$. Hence, $\sigma = d_1 \cdots d_n \leq dir(\tau)$. Since after $t = 2n$ clocks x and y are synchronous, we obtain $dir(\tau) = \sigma \cdot 2^\omega$. □

With the above lemma, we now prove Parts (3) and (4) of the theorem. Let $k > 1$ and $\ell > 0$. First, we will show that checking emptiness is undecidable. Given a PCP instance as before, we again consider the icTA (or DTA) $\mathcal{B}$ from Figure 7.9. We want to show that $w \in A^+$ is solution iff $wb \in \mathcal{L}_\forall(\mathcal{B}) = \mathcal{L}_\forall^{\text{rat},k}(\mathcal{B}) = \mathcal{L}_\forall^{\text{diff},\ell}(\mathcal{B})$. One direction is trivial. If, for $w \in A^+$, we have $f(w) = g(w)$, then $wb \in \mathcal{L}_\forall(\mathcal{B})$, and this implies that $wb \in \mathcal{L}_\forall^{\text{rat},k}(\mathcal{B})$ and $wb \in \mathcal{L}_\forall^{\text{diff},\ell}(\mathcal{B})$. On the other hand, if $wb \in \mathcal{L}_\forall^{\text{rat},k}(\mathcal{B})$ or $wb \in \mathcal{L}_\forall^{\text{diff},\ell}(\mathcal{B})$, then, by Lemma 7.20, we pick $\tau \in Rates_{\text{rat}}(k) \cap Rates_{\text{diff}}(\ell)$ such that $dir(\tau) = f(w) \cdot 2 \cdot 2^\omega$, and the remaining part of the proof follows as before.

Now, to show that the universality is undecidable, we consider icTA $\widetilde{\mathcal{B}}$ from Figure 7.11 and show that $w \in A^+$ is a solution iff wb is not in $\mathcal{L}_\forall^{\text{rat},k}(\widetilde{\mathcal{B}})$ or not in $\mathcal{L}_\forall^{\text{diff},\ell}(\widetilde{\mathcal{B}})$. One direction is again easy. If $wb \notin \mathcal{L}_\forall^{\text{rat},k}(\widetilde{\mathcal{B}})$ or $wb \notin \mathcal{L}_\forall^{\text{diff},\ell}(\widetilde{\mathcal{B}})$ then $wb \notin \mathcal{L}_\forall(\widetilde{\mathcal{B}})$ and since $wb \in A^+b$, we deduce that $f(w) = g(w)$ as in the

proof of Theorem 7.14. On the other hand, assume w is a solution so that we have $f(w) = g(w)$. Again by Lemma 7.20, we pick $\tau \in \mathit{Rates}_{\mathrm{rat}}(k) \cap \mathit{Rates}_{\mathrm{diff}}(\ell)$ such that $\mathit{dir}(\tau) = f(w) \cdot 2 \cdot 2^{\omega}$ and so by Claim 7.15, $(\widetilde{\mathcal{B}}, \tau) : s_0 \overset{wb}{\nrightarrow} r_1$ and $(\widetilde{\mathcal{B}}, \tau) : s_0 \overset{wb}{\nrightarrow} r_2$. We also have that after wb, states s_0, u_1,u_2 cannot be reached and so $wb \notin \mathcal{L}_{\forall}^{\mathrm{rat},k}(\widetilde{\mathcal{B}})$ and $wb \notin \mathcal{L}_{\forall}^{\mathrm{diff},\ell}(\widetilde{\mathcal{B}})$.

This completes the proof of the whole theorem.

□

As a related question, we could also ask if the existential semantics still describes a regular set of behaviours, i.e, for each $k \geq 1, \ell \geq 0$ are $\mathcal{L}_{\exists}^{\mathrm{rat},k}(\mathcal{B}), \mathcal{L}_{\exists}^{\mathrm{diff},\ell}(\mathcal{B})$ regular? We leave this as an open question. We note however, that this does not immediately follow from the region construction, since the restriction induced by the bounds may not result in classical zones (union of regions).

7.4.2 Restricting to fixed slopes

In this subsection, we restrict the behaviour by considering a selected subclass of local-time rate functions, rather than all of them. In particular, we restrict to the class of local-time rate functions that have fixed and constant (rational) slopes. Surprisingly, even checking emptiness of the existential semantics turns out to be undecidable with this restriction.

In fact, we show that checking emptiness of the existential semantics is undecidable even in a slightly weaker setting, where the local-time rate functions have a fixed slope with respect to each other. To see this, let us define $\mathit{Rates}_{\mathrm{fix}} = \{\tau \in \mathit{Rates} \mid \text{ for each pair } p, q \in \mathit{Proc}, \text{ there is a constant } \alpha_{pq} \in \mathbb{Q}_{\geq 0} \text{ such that } \tau_p(t) = \alpha_{pq} \cdot \tau_q(t) \text{ for all } t \in \mathbb{R}_{\geq 0}\}$. Again, for an icTA or DTA $\mathcal{B}$, we define $\mathcal{L}_{\exists}^{\mathrm{fix}}(\mathcal{B}) = \bigcup_{\tau \in \mathit{Rates}_{\mathrm{fix}}} \mathcal{L}(\mathcal{B}, \tau)$ and $\mathcal{L}_{\forall}^{\mathrm{fix}}(\mathcal{B}) = \bigcap_{\tau \in \mathit{Rates}_{\mathrm{fix}}} \mathcal{L}(\mathcal{B}, \tau)$. Now, we can state the theorem formally:

Theorem 7.21. *For* icTAs *or DTAs $\mathcal{B}$ over Proc, checking emptiness of $\mathcal{L}_{\exists}^{\mathrm{fix}}(\mathcal{B})$ is undecidable.*

The proof is by reducing to the above problem a certain "unknown-sampling rate discrete-time reachability problem" for timed automata [9, 22], which was proved in [22] to be undecidable. The idea is that the fixed constant β between the local-time rates of two processes can be used to simulate a β-sampled run of a timed automaton. To make this clear, we need to define sampled runs and the results known about them.

Recall that a timed run $\sigma = (a_1, t_1) \ldots (a_n, t_n)$, with $a_i \in \Sigma_{\varepsilon}, t_i \in \mathbb{R}_{\geq 0}$, of a timed automaton $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, I, i, F)$ can be seen as an alternating sequence of discrete moves and time elapse moves. Then, σ is called a *β-sampled run* for some $\beta \in \mathbb{Q}_{\geq 0}$, if each time elapse is exactly β, i.e., $t_i - t_{i-1} = \beta$ for all $i \in \{1, \ldots, n\}$,

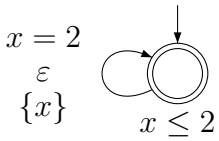


Figure 7.14: The timed automaton $\mathcal{B}_p$

$t_0 = 0$. A state s of $\mathcal{B}$ is reachable in the β-sampled semantics of $\mathcal{B}$, if there is a β-sampled run of $\mathcal{B}$ that reaches it. It is easy to see that this definition coincides with the one in [22] from which we have the following theorem:

Theorem 7.22 (cf. [22])**.** *The following problem is undecidable: Given a timed automaton $\mathcal{B}$ and a state s of $\mathcal{B}$, does there exist $\beta \in \mathbb{Q}_{\geq 0}$ such that s is reachable in the β-sampled semantics of $\mathcal{B}$.*

Proof. (of Theorem 7.21) Given a timed automaton $\mathcal{B}$ and a state s of $\mathcal{B}$, we will construct a DTA $\mathcal{D}$ such that $\mathcal{L}_{\exists}^{\text{fix}}(\mathcal{D}) \neq \emptyset$ if and only if there is a β such that s is reachable in the β-sampled semantics of $\mathcal{B}$. Thus, from the undecidability result of the theorem above, it follows that checking emptiness of $\mathcal{L}_{\exists}^{\text{fix}}(\mathcal{D})$ is undecidable.

The DTA we construct will consist of two components/processes. The broad idea is that the first component just measures its local time by making a transition at every clock tick. The second component simulates the timed automaton $\mathcal{B}$ and in addition checks a clock of the first component, to ensure that its transitions occur exactly at clock ticks of the first component. Thus, if the relative local-time rate is β, then a run of the automaton in the second component corresponds to a β-sampled run of the timed automaton.

There is a point of subtlety here. The automaton in the second component must make a transition every clock tick and cannot be allowed to remain at a state as time passes. It turns out that straightforward use of state invariants is not enough. Our construction below describes one way to handle these concerns.

Let $\mathcal{B} = (S_{\mathcal{B}}, \Sigma_{\mathcal{B}}, \mathcal{Z}_{\mathcal{B}}, \delta_{\mathcal{B}}, \mathit{Inv}_{\mathcal{B}}, \iota_{\mathcal{B}}, F_{\mathcal{B}})$ be the timed automaton and $s_{\mathcal{B}} \in S_{\mathcal{B}}$. We set $\mathit{Proc} = \{p, q\}$ and define DTA $\mathcal{D} = ((\mathcal{B}_p, \mathcal{B}_q), \pi)$ as follows.

The p-component uses and owns a single clock x (which occurs in both components, as will be seen). Thus, $\mathcal{Z}_p = \{x\}$ and $\pi(x) = p$. This automaton $\mathcal{B}_p$ is described in Figure 7.14.

The second component consists of two copies of $\mathcal{B}$ with transitions alternating between them, defined as follows: $\mathcal{B}_q = (S_q, \Sigma_q, \mathcal{Z}_q, \delta_q, \mathit{Inv}_q, \iota_q, F_q)$, where

- $S_q = S_{\mathcal{B}} \times \{0, 1\}$,
- $\Sigma_q = \Sigma_{\mathcal{B}}$,
- $\mathcal{Z}_q = \mathcal{Z}_{\mathcal{B}} \uplus \{x\}$ and $\pi(y) = q$ for all $y \in \mathcal{Z}_{\mathcal{B}}$,

- if $(s, \varphi, a, R, s') \in \delta_{\mathcal{B}}$, then $((s,0), \varphi, a, R, (s',1)), ((s,1), a, \varphi, R, (s',0)) \in \delta_q$,
- $Inv_q((s,0)) = Inv_{\mathcal{B}}(s) \wedge (0 \leq x \leq 1)$ and $Inv_q((s,1)) = Inv_{\mathcal{B}}(s) \wedge (1 \leq x \leq 2)$,
- $\iota_q = (\iota_{\mathcal{B}}, 0)$, and
- $F_q = \{(s_{\mathcal{B}}, 0), (s_{\mathcal{B}}, 1)\}$.

Thus, in the above construction of $\mathcal{B}_q$, in any state of the first copy, the value of clock x must be between 0 and 1. Similarly, in any state of the second copy, the clock has a value between 1 and 2. Further, transitions can only occur from one copy to another (never within a copy). These properties ensure that transitions (of $\mathcal{B}_q$) occur at every clock tick (of clock x) and there must occur a transition at every clock tick (else a state invariant is violated). But clock x measures its local time, i.e, the local time of process p. Thus, if β is the ratio between the fixed local-time rates of the two processes $\tau = (\tau_p, \tau_q) \in Rates_{\text{fix}}$, then each transition of $\mathcal{B}_q$ (which simulates $\mathcal{B}$) occurs at β-intervals, thus simulating a β-sampled run of $\mathcal{B}$. Finally, a run of $\mathcal{D}$ under this local-time rate function τ is accepting, if and only if the corresponding β-sampled run reaches $s_{\mathcal{B}}$. Thus, the existential fixed-slope semantics of $\mathcal{D}$ is not empty if and only if there is a β such that $s_{\mathcal{B}}$ is reachable in the β-sampled semantics. And further, the universal fixed-slope semantics of $\mathcal{D}$ is empty if and only if there is a β such that $s_{\mathcal{B}}$ is not reachable in the β-sampled semantics. This completes the proof of the theorem. □

We could also consider the emptiness problem for the universal semantics, i.e., is $\mathcal{L}^{\text{fix}}_{\forall}(\mathcal{D}) = \emptyset$? From the above construction, we have $\mathcal{L}^{\text{fix}}_{\forall}(\mathcal{D}) = \emptyset$ if and only if there is a β such that s is *not* reachable in the β-sampled semantics of $\mathcal{B}$. Thus if the safety problem for β-sampled semantics is undecidable then this would imply undecidability here as well.

7.5 The reactive semantics

The universal semantics described in the previous section is a possible way to implement positive specifications, i.e, to make sure that our system must satisfy some behavior irrespective of the time/clock evolution. Unfortunately, since emptiness and universality are undecidable even for bounded restrictions, it is not of any practical use. We would indeed like a semantics that describes only regular behaviors.

There is another subtle but powerful reason for looking for other semantics. When we want to check if the system satisfies a positive specification, we would like to be able to design a controller which can actually do this. For this, the semantics has to be "reactive". The universal semantics fails in this, in the sense

that, to choose a correct run in the system, we might need to know the future time rates.

Let us illustrate this problem through an example. Consider the icTA $\mathcal{B}$ from Figure 7.3. We have that ab is in the universal language since for any time rate, we can find an accepting run. In such an accepting run on ab, we start at s_0 and we can move to either s_1 or s_2. But this choice of which state to move to now (i.e, at s_0 with clock values $x < 1, y < 1$), depends on how the local time rates behave later. In particular, suppose we move to s_1, then it is possible that when $y = 1$, we still have $x < 1$, in which case, the transition reading b will never be enabled and we will be stuck. Similarly, if we move to s_2, then the local times might evolve in such a way that we have $y < 1$ and $x = 1$ and then we would be stuck at s_2. The universal semantics works by assuming that we can guess the whole future time rate function at any point. Thus, for instance, if we knew that later the region $y \leq 1, x \geq 1$ will be reached, then we would go to s_1 else we would go to s_2. (This can be seen pictorially from Figure 7.15.)

However, if we are designing a controller, then we do not know how the time rates will change later. Thus, we do not know apriori whether we will reach this region or the other, and so the controller actually has no way of deciding which transition to take, to ensure an accepting run. Thus, we can argue that in this case, though ab is in the universal language, there is no step-by-step, "practically constructible" accepting run on ab!

In this section, we introduce a new game-like semantics that solves both the above mentioned worries. It is regular and it is "reactive". We believe that this is the most promising semantics for checking positive specifications since it allows us to control the behaviour rather than guess it in advance. Formally, we will describe it using an alternating automaton, which is based on the region automaton introduced in Section 7.2. Intuitively, *time-elapse* transitions are controlled by the environment whereas *discrete* transitions are controlled by the system that aims at exhibiting some behavior. This *game* is not *turn-based* because the system should be able to execute several discrete transitions while staying in the same region. After moving from some region to a successor region, the environment hands over the control to the system so that the system always has a chance to execute some discrete transition. On the other hand, after executing some discrete transition, the system may either keep the control or hand it over to the environment.

As suggested, our reactive semantics will be described by alternating automata. Since icTAs or DTAs have ε-transitions, we define an *alternating automaton with ε-transitions* (ε-AA) as a tuple $\mathcal{A} = (S, \Sigma, \delta, \iota, F)$ where S is a finite set of *states*, $\iota \in S$ is the *initial state*, $F \subseteq S$ is the set of *final states*, and $\delta : S \times \Sigma_\varepsilon \to \mathbb{B}^+(S)$ is the *alternating transition function*. Here, $\mathbb{B}^+(S)$ denotes positive boolean combinations of states from S.

As usual, a run of an ε-AA will be a (doubly) labeled finite tree. We assume the reader to be familiar with the notion of trees and only mention that we deal

with structures (V, σ, μ) where V is the finite set of nodes with a distinguished root, and both σ and μ are node-labeling functions. Given a node $u \in V$, the set of children of u is denoted children(u). Let $w = a_1 \dots a_{|w|} \in \Sigma^*$ be a finite word. A run of $\mathcal{A}$ on w is a doubly labeled finite tree $\rho = (V, \sigma, \mu)$ where $\sigma : V \to S$ is the *state-labeling* function and $\mu : V \to \{0, \dots, |w|\}$ is the *position-labeling* function such that, for each node $u \in V$, the following hold:

- if u is the root, then $\sigma(u) = \iota$ and $\mu(u) = 0$ (we start in the initial state at the beginning of the word),
- if u is not a leaf (i.e., children$(u) \neq \emptyset$), then we have
  - either $\mu(u') = \mu(u)$ for all $u' \in$ children(u) and in this case $\{\sigma(u') \mid u' \in \text{children}(u)\} \models \delta(\sigma(u), \varepsilon)$
  - or $\mu(u') = \mu(u) + 1 = i \leq n$ for all $u' \in$ children(u) and in this case $\{\sigma(u') \mid u' \in \text{children}(u)\} \models \delta(\sigma(u), a_i)$.

The run is accepting if all leaves are labeled with $F \times \{|w|\}$. The set of words from Σ^* that come with an accepting run is denoted by $\mathcal{L}(\mathcal{A})$. This set is indeed regular since ε-AA's are special cases of two-way alternating automata (2-AA) which accept only regular languages.

Lemma 7.23 (cf. [15])**.** *Given a 2-AA $\mathcal{A}$ with n states, one can construct a non-deterministic finite automaton with $2^{\mathcal{O}(n^2)}$ states that recognizes $\mathcal{L}(\mathcal{A})$.*

Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, \mathit{Inv}, \iota, F, \pi)$ be an icTA over *Proc*. We associate with $\mathcal{B}$ an ε-AA $\mathcal{A}_\mathcal{B} = (S', \Sigma, \delta', \iota', F')$ as follows: First, let $S' = S \times \mathit{Regions}(\mathcal{B}) \times \{0, 1\}$. Intuitively, tag 0 is for *system positions* while tag 1 is for *environment positions* (recall that the environment controls how time elapses whereas the system wants to accept some word). Then, $\iota' = (\iota, [\nu], 0)$ where $\nu(x) = 0$ for each $x \in \mathcal{Z}$, and $F' = F \times \mathit{Regions}(\mathcal{B}) \times \{0, 1\}$. Finally, for $(s, \gamma) \in S \times \mathit{Regions}(\mathcal{B})$ and $a \in \Sigma_\varepsilon$, we let

$$
\begin{array}{rcl}
\delta'((s,\gamma,1),a) & = & \text{False} \quad \text{if } a \neq \varepsilon \\
\delta'((s,\gamma,1),\varepsilon) & = & \bigwedge\{(s,\gamma',0) \mid \gamma \prec \gamma'\} \\
\delta'((s,\gamma,0),a) & = & \begin{cases} \bigvee\{(s',\gamma',0) \mid (s,\gamma) \xrightarrow{a}_{\mathrm{d}} (s',\gamma')\} & \text{if } a \neq \varepsilon \text{ or } \gamma \text{ maximal} \\ (s,\gamma,1) \vee \bigvee\{(s',\gamma',0) \mid (s,\gamma) \xrightarrow{\varepsilon}_{\mathrm{d}} (s',\gamma')\} & \text{otherwise} \end{cases}
\end{array}
$$

where $\xrightarrow{a|\varepsilon}_{\mathrm{d}}$ denotes a discrete transition of the region automaton $\mathcal{R}_\mathcal{B}$ (Section 7.2).

Definition 7.24. *For an* icTA $\mathcal{B}$*, let $\mathcal{L}_{react}(\mathcal{B}) = \mathcal{L}(\mathcal{A}_\mathcal{B})$ be the* reactive semantics *of $\mathcal{B}$. Moreover, for a DTA $\mathcal{D}$, $\mathcal{L}_{react}(\mathcal{D}) = \mathcal{L}_{react}(\mathcal{B}_\mathcal{D})$ is the* reactive semantics *of $\mathcal{D}$.*

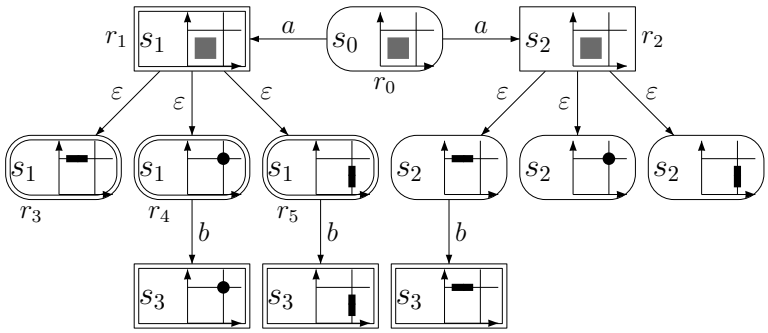


Figure 7.15: Part of the region/alternating automaton for the icTA from Figure 7.3

Example 40. Consider, again, the icTA $\mathcal{B}$ from Figure 7.3. A part of its ε-AA $\mathcal{A}_\mathcal{B}$ is shown in Figure 7.15. States with tag 0 are depicted as ovals and are existential (non-deterministic) states and states with tag 1 are depicted as rectangles and are universal states. We have, e.g., $\delta'(r_1, \varepsilon) = r_3 \wedge r_4 \wedge r_5$. Note, however, that a transition from an oval to a rectangles should actually be split into two transitions, which is omitted in the picture. For example, there is a state r_1' between r_0 and r_1 which resembles r_1 but is tagged 0. Similarly, there is another state r_2' between r_0 and r_2, and we have $\delta'(r_0, a) = r_1' \vee r_2'$. Then, as mentioned in the beginning of the section, under the reactive semantics, the language of this automaton contains a but does not contain ab. Thus, $\mathcal{L}_{react}(\mathcal{B}) = \{a\}$.

The following theorem follows from Lemma 7.23:

Theorem 7.25. *Let $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ be an* icTA *and let n be the number of states of $\mathcal{R}_\mathcal{B}$ (which is bounded by $|S| \cdot (2\,C + 2)^{|\mathcal{Z}|} \cdot |\mathcal{Z}|!$ where C is the largest constant a clock is compared with in $\mathcal{B}$). Then, $\mathcal{L}_{react}(\mathcal{B})$ is regular and one can compute a non-deterministic finite automaton with $2^{\mathcal{O}(n^2)}$ states that recognizes $\mathcal{L}_{react}(\mathcal{B})$.*

As expected, the reactive semantics is more restricted than the universal semantics, so we get the inclusion of Proposition 7.26. Therefore, we can safely use the reactive semantics to check an icTA for a positive specification *Good* containing the behaviors that a system *must* exhibit. If $Good \subseteq \mathcal{L}_{react}(\mathcal{B})$ then the system has a strategy to ensure each *good* behaviors robustly against all possible clock variations.

Proposition 7.26. *For any* icTA $\mathcal{B}$, $\mathcal{L}_{react}(\mathcal{B}) \subseteq \mathcal{L}_\forall(\mathcal{B})$.

Proof. Assume $\mathcal{B} = (S, \Sigma, \mathcal{Z}, \delta, Inv, \iota, F, \pi)$ to be an icTA and let furthermore $\mathcal{A}_\mathcal{B} = (S', \Sigma, \delta', \iota', F')$ be the associated ε-AA, and let $w \in \mathcal{L}_{react}(\mathcal{B}) = \mathcal{L}(\mathcal{A}_\mathcal{B})$. Let furthermore $\rho = (V, \sigma, \mu)$ be an accepting run of $\mathcal{A}_\mathcal{B}$ on w. We pick $\tau \in Rates$.

We construct inductively a maximal branch $u_0 u_1 \dots u_n \in V^*$ in ρ and two sequences $t_0, t_1, \dots, t_n$ and $\nu_0, \nu_1, \dots, \nu_n$ as follows. First, we let $u_0 = \varepsilon$, $t_0 = 0$

and $\nu_0(x) = 0$ for all $x \in \mathcal{Z}$. Note that $\sigma(u_0) = (\iota, [\nu_0], 0)$. Assume that the sequences have been constructed up to k and that $\sigma(u_k) = (s_k, [\nu_k], pl_k)$. If u_k is a leaf, the construction is over and $n = k$. Otherwise, there are three cases. First, assume that $pl_k = 1$. Let $t_{k+1} > t_k$ be such that $[\nu_k] \prec [\nu_{k+1}]$ with $\nu_{k+1} = \nu_k + \tau(t_{k+1}) - \tau(t_k)$. By definition of δ', there exists a child u_{k+1} of u_k such that $\sigma(u_{k+1}) = (s_k, [\nu_{k+1}], 0)$. Assume now that $pl_k = 0$. Choose u_{k+1} in children(u_k). Either $\sigma(u_{k+1}) = (s_k, [\nu_k], 1)$ and we let $t_{k+1} = t_k$ and $\nu_{k+1} = \nu_k$ in this second case. Otherwise, the move from u_k to u_{k+1} corresponds to some discrete transition of $\mathcal{R}_\mathcal{B}$ with label $a_{k+1} \in \Sigma_\varepsilon$ and some reset set $R \subseteq \mathcal{Z}$. In this third case, we let $t_{k+1} = t_k$ and $\nu_{k+1} = \nu_k[R]$ so that we have $\sigma(u_{k+1}) = (s_{k+1}, [\nu_{k+1}], 0)$.

The discrete moves along the constructed branch correspond to the sequence $0 < i_1 < \cdots < i_\ell \leq n$ of indices k such that $pl_{k-1} = pl_k = 0$. As ρ is an accepting run for w, we have $w = a_{i_1} \cdot a_{i_2} \cdot \cdots \cdot a_{i_\ell}$ and $s_{i_\ell} = s_n \in F$. It is now easy to verify that the sequence

$$(s_0, \nu_0) \xrightarrow{a_{i_1}, t_{i_1}} (s_{i_1}, \nu_{i_1}) \xrightarrow{a_{i_2}, t_{i_2}} \dots \xrightarrow{a_{i_\ell}, t_{i_\ell}} (s_{i_\ell}, \nu_{i_\ell})$$

is a τ-run of $\mathcal{B}$ so that $w \in \mathcal{L}(\mathcal{B}, \tau)$. □

To summarize, we have the following strict hierarchy of our semantics.

Proposition 7.27. *Suppose that $|Proc| \geq 2$. There are some DTA $\mathcal{D}$ over Proc and some $\tau \in Rates$ such that $\mathcal{L}_{react}(\mathcal{D}) \subsetneqq \mathcal{L}_\forall(\mathcal{D}) \subsetneqq \mathcal{L}(\mathcal{D}, \tau) \subsetneqq \mathcal{L}_\exists(\mathcal{D})$.*

Proof. Consider the icTA $\mathcal{B}$ from Figure 7.3. Recall that $\mathcal{L}_{react}(\mathcal{B}) = \{a\}$, $\mathcal{L}_\forall(\mathcal{B}) = \{a, ab\}$, $\mathcal{L}(\mathcal{B}, \mathrm{id}) = \{a, ab, b\}$, and $\mathcal{L}_\exists(\mathcal{B}) = \{a, ab, b, c\}$. As $\mathcal{B}$ does not employ any reset, we may view it as a DTA where $\mathcal{B}$ models a process owning clock x, and where a second process, owning clock y, does nothing, but is in a local accepting state. □

8

Conclusions and Future Work

In this thesis, we have used formal methods to study and model systems where the interplay of time and concurrency is crucial to the behaviour of the system.

In the first part of the thesis, we have focussed on using timed partial orders as formalisms to describe the behaviours of such systems. In Chapter 3 we began by introducing these formalisms, i.e., TMSCs and TCMSCs. Then we defined two automata models that exhibit such behaviour, i.e., TMPA and ECMPA. Finally, we introduced the TCMSG formalism that is used to specify collections of such behaviours.

The first main result proved in Chapter 4 shows that timed monadic second-order logic formulae and ECMPA are expressively equivalent over TMSCs. Indeed, this equivalence holds without assumptions on the bounds of channels, only when we restrict to the existential fragment of the logic. Further, the proof of this equivalence is constructive, since we are able to formulate an explicit translation between the two. The second main result in Chapter 5 proves that checking emptiness for ECMPA is decidable in the bounded case. These two results together allow us to check satisfiability for the timed logic. These results have been published in [2].

In Chapter 6 we have considered some problems that arise when checking if the specification (given as a TCMSG) conforms with the implementation (given as a TMPA). The solutions are provided in a natural setting, where we assume the TCMSG to be locally synchronized. We have first shown that a locally synchronized TCMSG recognizes a timed regular language. Then, we improved this result by showing that the automaton that we construct to demonstrate regularity can be determinized and hence complemented. This allowed us to provide a solution to the consistency problem in Section 6.5. Finally, with the additional assumption that TCMSGs are event-saturated, we provided a solution for the coverage problem stated in Section 6.6. A preliminary version of some of these results appeared in [4].

In the second part of the thesis, we have tried to focus on a different phenomenon that is interesting in the context of combining time and concurrency. This concerns the notion of local time obtained by allowing independently evolving clocks. We have introduced a distributed timed automaton model, DTA, where

clocks on different processes can evolve at rates that are independent of one another. the interaction between processes is captured by allowing processes to read the clock values of one another. We have also defined an intermediate global model icTA, where the states are shared but the clocks still evolve independently.

We defined the existential and universal semantics to capture the behaviours that may occur under some choice of clock rates and the behaviours that must occur under any choice of clock rates, respectively. We proved that the former always generates a regular set of behaviours. The universal semantics however turned out to be intractable since checking emptiness was shown to be undecidable. This result was then strengthened to show undecidability of the semantics even when the time rates on different processes are not arbitrary but have some fixed relationship. Finally, to be able to guarantee behaviour under any choice of clock rates, we introduced a game-like reactive semantics. We were able to show that this always yields a regular set of behaviours by constructing an equivalent alternating automaton. This work has appeared as a preliminary version in [3].

Future work and perspectives We see this thesis as a starting point for a better understanding of timed and distributed systems in a formal setting. There are several levels at which one could branch out and explore extensions of the work reported here.

At a very immediate level, we would like to improve some of the results for problems that we have already introduced in the thesis. For instance, some of our results are proved under natural assumptions and we would be interested in how to extend them to more general settings. In particular, it would be nice to relax some of the restrictions on the TCMSG (event-saturated, locally-synchronized), that were required in our current approach in Chapter 6. Further, in Section 7.4, when we consider DTA with fixed slopes or bounded drifts, there are some cases where we have not settled the decidability (or indeed, the undecidability) of the problem. These are technical issues that nevertheless provide valuable intuition towards understanding the model better.

At the next level, there are several well-defined open problems that arise naturally from our models and results. We plan to investigate the expressive power of DTAs and, in particular, the *synthesis problem*: For which (global) specifications *Spec* can we generate a DTA $\mathcal{D}$ (over some given system architecture) such that $\mathcal{L}_{react}(\mathcal{D}) = Spec$? A similar synthesis problem has been studied in [38] in the framework of untimed distributed channel systems. There, additional messages are employed to achieve a given global behavior. In this context, it would be helpful to have partial-order based specification languages and a partial-order semantics for DTAs (see, for example, [50]).

At a higher level, it would be interesting to relate our approach for modelling timed distributed systems to other formalisms that were mentioned in the intro-

duction. As one such instance, we would like to compare and contrast unfoldings of time petri nets with our approach. Also, it would be interesting to see what happens if we replace timed automata by hybrid automata while considering independently evolving clocks.

Finally, we hope that the work in this thesis, by leading to a better understanding of timed and distributed systems, will eventually find application in the design and analysis of the systems that surround us.

Bibliography

[1] S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. Research Report LSV-08-19, ENS Cachan, 2008.

[2] S. Akshay, Benedikt Bollig, and Paul Gastin. Automata and logics for timed message sequence charts. In *FSTTCS*, pages 290–302, 2007.

[3] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In *CONCUR*, pages 82–97, 2008.

[4] S. Akshay, Madhavan Mukund, and K. Narayan Kumar. Checking coverage for infinite collections of timed scenarios. In *CONCUR*, pages 181–196, 2007.

[5] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.

[6] R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *TCS*, 211(1-2):253–273, 1999.

[7] R. Alur, G. Holzmann, and Doron Peled. An analyser for message sequence charts. In *Proc. of TACAS 1996*, pages 35–48, 1996.

[8] R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. of CONCUR 1999*, volume 1664 of *LNCS*, pages 114–129. Springer, 1999.

[9] Rajeev Alur and P. Madhusudan. Decision problems for timed automata: A survey. In *SFM*, pages 1–24, 2004.

[10] H. Ben-Abdallah and S. Leue. Timing constraints in message sequence chart specifications. In *Proc. of FORTE 1997*, pages 91–106, 1997.

[11] J. Bengtsson, B. Jonsson, J. Lilius, and W. Yi. Partial order reductions for timed systems. In *Proc. of CONCUR'98*, pages 485–500. Springer, 1998.

[12] J. Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *Lectures on Concurrency and Petri Nets 2003*, volume 3098 of *LNCS*, pages 87–124. Springer-Verlag, 2003.

[13] B. Bérard, V. Diekert, P. Gastin, and A. Petit. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2):145–182, 1998.

[14] Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Software Eng.*, 17(3):259–273, 1991.

[15] J.-C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26(3):237–269, 1993.

[16] B. Bollig. *Automata and logics for message sequence charts*. PhD thesis, Lehrstuhl für Informatik II, RWTH Aachen, 2005.

[17] B. Bollig and M. Leucker. Message-passing automata are expressively equivalent to EMSO logic. *TCS*, 358(2-3):150–172, 2006.

[18] P. Bouyer, S. Haddad, and P.-A. Reynier. Timed unfoldings for networks of timed automata. In *Proc. of ATVA'06*, volume 4218 of *LNCS*. Springer, 2006.

[19] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2), 1983.

[20] J. Büchi. Weak second order logic and finite automata. *Z. Math. Logik, Grundlag. Math.*, 5:66–62, 1960.

[21] F. Cassez, Th. Chatain, and C. Jard. Symbolic unfoldings for networks of timed automata. In *Proc. of ATVA'06*, volume 4218 of *LNCS*, pages 307–321. Springer, 2006.

[22] F. Cassez, T. A. Henzinger, and J.-F. Raskin. A comparison of control problems for timed and hybrid systems. In *Proc. of HSCC'02*, volume 2289 of *LNCS*, pages 134–148. Springer, 2002.

[23] Joy Chakraborty, Deepak D'Souza, and K. Narayan Kumar. Analysing message sequence graph specifications. Technical Report IISc-CSA-TR-2009-1, IISc Bangalore, 2009.

[24] P. Chandrasekaran and M. Mukund. Matching scenarios with timing constraints. In *Proc. of FORMATS 2006*, pages 91–106, 2006.

[25] Thomas Chatain. *Dépliages symboliques de réseaux de Petri de haut niveau et application à la supervision des systèmes répartis*. Thèse de doctorat, Université Rennes 1, Rennes, France, November 2006.

[26] Thomas Chatain and Claude Jard. Time supervision of concurrent systems using symbolic unfoldings of time petri nets. In *FORMATS*, pages 196–210, 2005.

[27] Mireille Clerbout and Michel Latteux. Semi-commutations. *Information and Computation*, 73(1):59–74, 1987.

[28] R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106:159–202, 1993.

[29] M. De Wulf, L. Doyen, N. Markey, and J.-F. Raskin. Robustness and implementability of timed automata. In *Proc. of FORMATS'04 and FTRTFT'04*, volume 3253 of *LNCS*, pages 118–133. Springer, 2004.

[30] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.

[31] C. Dima and R. Lanotte. Distributed time-asynchronous automata. In *Proc. of ICTAC'07*, volume 4711 of *LNCS*. Springer, 2007.

[32] L. Doyen. *Algorithmic Analysis of Complex Semantics for Timed and Hybrid Automata*. PhD thesis, Université Libre de Bruxelles, 2006.

[33] D. D'Souza. *A Logical Study of Distributed Timed Automata*. PhD thesis, BITS Pilani, 2000.

[34] D. D'Souza. A logical characterisation of event clock automata. *International Journal of Foundations of Computer Science*, 14(4):625–640, 2003.

[35] D. D'Souza and P. S. Thiagarajan. Product interval automata: A subclass of timed automata. In *Proc. of FSTTCS 1999*, pages 60–71. Springer-Verlag, 1999.

[36] Deepak D'Souza and Madhavan Mukund. Checking consistency of sdl+msc specifications. In *SPIN*, pages 151–165, 2003.

[37] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–52, 1961.

[38] B. Genest. On implementation of global concurrent systems with local asynchronous controllers. In *Proc. of CONCUR'05*, volume 3653 of *LNCS*, pages 443–457, 2005.

[39] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Information and Computation*, 204(6):920–956, 2006.

[40] Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems - An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.

[41] O. Grinchtein and M. Leucker. Network invariants for real-time systems. *Formal Aspects of Computing*, 2008. to appear.

[42] J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Information and Computation*, 202(1):1–38, 2005.

[43] T. A. Henzinger. The theory of hybrid automata. In *Proc. of LICS'96*, 1996.

[44] ITU-TS Recommendation Z.120anb: Formal Semantics of Message Sequence Charts, 1998.

[45] ITU-TS Recommendation Z.120: Message Sequence Chart 1999 (MSC99), 1999.

[46] P. Krcal and W. Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *Proc. of CAV 2006*, volume 4144 of *LNCS*, pages 243–257. Springer, 2006.

[47] K. Narayan Kumar. The theory of msc languages. *World Scientific Review Volume*, 2009. To appear.

[48] D. Kuske. Regular sets of infinite message sequence charts. *Information and Computation*, 187:80–109, 2003.

[49] K. G. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proc. of RTSS'95*, page 76. IEEE Computer Society, 1995.

[50] D. Lugiez, P. Niebert, and S. Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *TCS*, 345(1):27–59, 2005.

[51] A. Muscholl and D. Peled. Message sequence graphs and decision problems on mazurkiewicz traces. In *Proc. of MFCS 1999*, volume 1672 of *LNCS*, pages 81–91. Springer, 1999.

[52] Anca Muscholl, Doron Peled, and Zhendong Su. Deciding properties for message sequence charts. In *FoSSaCS*, pages 226–242, 1998.

[53] A. Puri. Dynamical properties of timed automata. *Discrete Event Dynamic Systems*, 10(1-2):87–113, 2000.

[54] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets*. PhD thesis, Massachusetts Institute of Technology, 1974.

[55] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *Monographs in Theoretical Computer Science. An EATCS Series*. Springer, 1985.

[56] M. Swaminathan, M. Fränzle, and J.-P. Katoen. The surprising robustness of (closed) timed automata against clock-drift. In *Proceedings of IFIP-TCS'08*. Springer, 2008. to appear.

[57] M. Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of MFCS'98*, volume 1443 of *LNCS*. Springer, 1998.

[58] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.