University of Namur

DOCTOR OF SCIENCES

Modeling and Verifying Distributed and Real-Time Systems using Timed Automata with Partially Independent Clocks

Ortiz Vega, James Jerson

Award date:
2023

Awarding institution:
University of Namur

Link to publication

Modeling and Verifying Distributed and Real-Time Systems using Timed Automata with Partially Independent Clocks

James Ortiz

Jury

Prof. S. Akshay
IIT-Bombay, India

Prof. Jean-Marie Jacquet
University of Namur, Belgium

Prof. Jean-François Raskin
Free University of Brussels, Belgium

Prof. Pierre-Yves Schobbens
University of Namur, Belgium

Prof. Stavros Tripakis
Northeastern University, USA

Prof. Wim Vanhoof
University of Namur, Belgium

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in the subject of Computer Science

Supervised by Prof. Pierre-Yves Schobbens

University of Namur
FOCUS Research Group / PReCISE Research Center

UNIVERSITÉ DE NAMUR
FACULTÉ D'INFORMATIQUE

© Presses universitaires de Namur & James Ortiz, 2023
Rue Grandgagnage, 19
B - 5000 Namur (Belgium)
pun@unamur.be - www.pun.be

Registration of copyright: D/2023/1881/12
ISBN: 978-2-39029-175-6
Printed in Belgium.

To my $\heartsuit^3$

“Anyone who has never made a mistake has never tried anything new.”

— ***Albert Einstein***

ABSTRACT

Distributed Real-Time Systems (DRTS) play an increasingly important role in everyday life, from traffic light controllers to airplanes, and from telecommunication networks to medical systems. In the last decades, several formal methods and real-time formalisms have been proposed to formalize and prove properties of DRTS. However, traditional real-time formalisms are not always adequate for reasoning about DRTS because they assume a unique, perfectly synchronous (Newtonian) measure of time. The most successful techniques for modeling real-time systems (RTS) are Timed Automata (TA) [AD94], Event Clock Automata (ECA) [AFH94], and Recursive Event Clock Automata (RECA) [HRS98]. A TA is a finite automaton augmented with real-valued clocks, where all clocks have infinite precision and are perfectly synchronized. This causes TA to have an undecidable language inclusion problem [AD94]. These negative results for TA spurred the search for expressive but still fully decidable formalisms. To restore decidability, Alur et al. [AFH94] proposed to restrict the behavior of clocks. Thus, an event clock (EC) is reset when a given atomic proposition occurs. The values of the event clocks are deterministic, and thus Event Clock Automata (ECA) are determinizable, which makes the language inclusion decidable.

However, the expressiveness of ECA is rather weak. Furthermore, the temporal logic with event clocks [RS97] violates the substitution principle: Every proposition should be replaceable by a formula. Therefore, Henzinger et al. [HRS98] introduced the notion of Recursive Event. In a recursive event model (RECA), the reset of a clock is decided by a lower-level automaton (or formula). This automaton cannot read the clock it is resetting. Clock resets are thus still deterministic, but the concept of event is now much more expressive. Also, Henzinger et al. [HRS98] introduced the temporal logic of recursive event clocks (EventClockTL).

There are other variants of TA called Distributed Timed Automata (DTA) and Timed Automata with Independent Clocks (icTA) proposed by Krishnan et al. [Kri99] and Akshay et al. [ABG$^+$08] to model DRTS where the clocks are not necessarily synchronized. Constraints on the clocks are used to restrict the behavior of the automata. In DTA, the clocks belonging to one process can be read by another process, but a clock can only be reset by its owner process. DTA and icTA are neither determinable nor complementable, and their inclusion problems are undecidable [ABG$^+$08]. However, the standard semantics of TA (DTA and icTA) is based on a Timed Labelled Transition System (TLTS), i.e. a run of a TA is given by a sequence of actions and a (single) global timestamp while using multiple clocks

and correspondingly associating multiple timestamps could be a more compact and functional representation of DRTS.

Moreover, it is a well-known fact that model checking over DRTS quickly becomes intractable because the state space often grows exponentially with the number of components considered. One of the most effective and successful techniques currently available for reducing the state space is to merge states with the same behavior. For untimed systems, the notion of bisimulation [Mil89] is classically used for this purpose, and its natural extension for RTS, timed bisimulation, has been used to verify the preservation of sequential behavior and timed properties expressed in modal and timed temporal logics, such as Timed Computational Temporal Logic (TCTL) [TY01] and L_v [LLW95]). Timed bisimulation has already been shown to be decidable for TA [AD94, LLW95], but its standard definition is also based on Timed Labelled Transition Systems (TLTS).

In this thesis, we remove the above problems of undecidability, perfect clock synchronization, large representation, and single timestamps. Here, inspired by [BJLY98, Kri99, ABG$^+$08, HRS98] and [AFH94], we introduce three alternative semantics:

(i) Distributed Event Clocks (DEC): A DEC x^q (or y^q) records the time since the last (or next) reset, measured in the local time of process q, and DECs can advance independently if they are in different processes. In contrast, Puri et al. [Pur98] and Wulf et al. [DWDMR04] studied the opposite case, where the difference between the clocks (drift) is infinitesimally small. In chapter 6 we extend the semantics of RECA and EventClockTL to include the notion of DEC. Therefore, we propose a formal semantics for modeling DRTS based on Recursive Event Clock Automata (RECA) with such distributed (a.k.a. independent) clocks, yielding the Distributed Recursive Event Clock Automata (DECA). We will show that DECA are determinizable, i.e. closed under complementation, and that their respective language inclusion problems are decidable (more precisely, PSPACE-complete). In addition, we propose to extend the existing timed temporal logic (EventClockTL) with distributed clocks to allow the specification of distributed and timed temporal properties. This gives us the Distributed (Recursive) Event Clock Temporal Logic (DECTL), which we show to be PSPACE-complete for the satisfiability and validity problems. Finally, we show the applicability of DECA and DECTL to a DRTS.

(ii) Multiple Independent Clocks (MIC): A DRTS involves multiple interconnected real-time processes (called components here), where each component uses its independent local clocks running at its own rate. Thus, a single timestamp is sufficient for global clocks, while multiple timestamps support independent local clocks. In chapter 7 we extend the standard semantics of TA and timed bisimulation to include the notion of multiple independent clocks.
Therefore, we extend the Timed Labelled Transition Systems (TLTS) and icTA semantics [ABG$^+$08] to work with the notion of MIC and multi-timed words. Therefore, we propose a new real-time formalism called Multi-timed Automata (MTA) based on icTA (and MIC). We also propose a timed modal logic with independent local clocks, which leads to the (multi-timed) modal logic ML_v. We

extend the classical theory of timed bisimulation [Cer93] with the new notion of multi-timed bisimulation and MTA. We show that multi-timed bisimulation is decidable (more precisely, EXPTIME-complete). Furthermore, we propose an efficient algorithm for multi-timed bisimulation using refinement techniques. Finally, we show the applicability of MTA and ML_v to a DRTS.

(iii) Distributed Clock Derivatives (DCD): A DCD $\dot{x}$ is a time derivative of the clock x. Comparisons can be made between two clock derivatives or between a clock derivative and a natural constant 1. In the chapter 8 we extend the semantics of TA to include the notion of distributed clock derivatives. Therefore, we extend the TA and L_v semantics [AD94] [LLW95] to work with the notion of DCD and multi-timed words, yielding the (derivative) multi-timed modal logic DML_v and Timed Automata with Clock Derivatives (DMTA). We show that (derivative) multi-timed bisimulation is decidable (more precisely, EXPTIME-complete). Furthermore, we propose an efficient algorithm for (derivative) multi-timed bisimulation using refinement techniques. Finally, we show the applicability of DMTA and DML_v to a DRTS.

Keywords: Formal Verification, Model Checking, Timed Automata, Timed Temporal Logics, Timed Bisimulation.

RÉSUMÉ

Les systèmes distribués en temps réel (SDTR) jouent un rôle de plus en plus important dans la vie quotidienne, des contrôleurs de feux de circulation aux avions et des réseaux de télécommunication aux systèmes médicaux. Au cours des dernières décennies, plusieurs méthodes formelles et formalismes du temps réel ont été proposés pour formaliser et prouver les propriétés de SDTR. Mais les formalismes traditionnels en temps réel ne sont pas toujours adéquats pour raisonner sûr SDTR, car ils supposent une mesure du temps unique et parfaitement synchrone (newtonienne). Les techniques les plus performantes pour modéliser les systèmes temps réel (STR) sont les automates temporisés (AT) [AD94], les automates d'horloges d'événements (AHE) [AFH94] et les automates récursifs d'horloges d'événements (ARHE) [HRS98]. Un AT est un automate fini augmenté d'horloges à valeurs réelles, où toutes les horloges ont une précision infinie et sont parfaitement synchronisées. Cela fait que AT ont un problème d'inclusion de langue indécidable [AD94]. Ces résultats négatifs pour TA ont stimulé une recherche fondamentale des formalismes expressifs, mais toujours entièrement décidables. Pour restaurer la décidabilité, Alur et al. [AFH94] a proposé de restreindre le comportement des horloges. Par conséquent, une horloge d'événement (HE) est réinitialisée lorsqu'une proposition atomique donnée se produit. Les valeurs d'horloges d'événements sont déterministes et donc les automates d'horloges d'événements (AHE) sont déterminables, ce qui rend l'inclusion du langage décidable.

Cependant, l'expressivité de AHE est plutôt faible. De plus, la logique temporelle avec d'horloges d'événements [RS97] viole le principe de substitution : toute proposition devrait être remplaçable par une formule. Ainsi, Henzinger et al. [HRS98] a introduit la notion d'Événement Récursif. Dans un modèle d'événements récursifs (ARHE), la remise à zéro d'une horloge est décidée par un automate (ou formule) de niveau inférieur. Cet automate ne peut pas lire l'horloge qu'il remet à zéro. Les réinitialisations d'horloge sont en conséquence toujours déterministes, mais le concept d'événement est maintenant beaucoup plus expressif. Aussi, Henzinger et al. [HRS98] a introduit la logique temporelle des horloges événementielles récursives (EventClockTL).

Il existe d'autres variantes de AT appelées automates temporisées distribuées (ATD) et automates temporisés avec horloges indépendantes (ATHI) qui ont été proposées par Krishnan et al. [Kri99] et Akshay et al. [ABG+08] pour modéliser SDTR, où les horloges ne sont pas nécessairement synchronisées. Des contraintes sur les horloges sont utilisées pour restreindre les comportements des automates. Dans

ATD, les horloges appartenant à un processus peuvent être lues par un autre processus, mais une horloge ne peut être réinitialisée que par son processus propriétaire. ATD et ATHI ne sont ni déterminables ni complémentables et leurs problèmes d'inclusion sont indécidables [ABG$^+$08]. Cependant, la sémantique standard de AT (ATD et ATHI) est basée sur un Système de transition étiquetée temporisée (STET), c'est-à-dire une exécution d'un AT est donné par une séquence d'actions et un (unique) horodatage global, tandis qu'utiliser plusieurs horloges et les associer en conséquence plusieurs horodatages pourrait être une représentation plus compacte et fonctionnelle du vrai SDTR.

De plus, c'est un fait bien connu que la vérification de modèle sur SDTR déviant rapidement insoluble, car l'espace d'états croît souvent de façon exponentielle avec le nombre de composants considérés. L'une des techniques les plus efficaces et les plus réussies actuellement disponibles pour réduire l'espace d'état consiste à fusionner des états ayant le même comportement. Pour les systèmes non temporisés, la notion de bisimulation [Mil89] est classiquement utilisée à cette fin, et son extension naturelle pour STR, la bisimulation temporisée, a été utilisée pour vérifier la préservation du comportement séquentiel et des propriétés temporisées exprimées en modal et logiques temporelles temporisées, telles que la logique temporelle de calcul temporisée (LTCT) [TY01] et L_v [LLW95]). La bisimulation temporisée s'est déjà avérée décidable pour AT [AD94, LLW95], mais sa définition standard est également basée sur les systèmes de transition étiquetés temporisés (STET).

Dans cette thèse, nous supprimons les problèmes ci-dessus de indécidabilité, synchronisation parfaite des horloges, grande représentation et horodatage unique. Ici, inspirés par [BJLY98, Kri99, ABG$^+$08, HRS98] et [AFH94], nous introduisons trois sémantiques alternatives :

(i) Horloges d'événements distribués (HED) : une HED x^q (ou y^q) enregistrée le temps écoulé depuis la dernière (resp. suivante) réinitialisation, mesuré dans le temps local du processus q et HED peuvent avancer de manière totalement indépendante s'ils sont dans des processus différents. En revanche, Puri et al. [Pur98] et Wulf et al. [DWDMR04] a étudié le cas inverse, où la différence entre les horloges (dérive) est infiniment petite. Dans le chapitre 6, nous étendons la sémantique de ARHE et EventClockTL afin d'inclure la notion de HED. Par conséquent, nous proposons une sémantique formelle pour la modélisation de SDTR basée sur des automates d'horloge à événements récursifs (ARHE) avec de telles horloges distribuées (a.k.a. indépendantes), donnant les automates d'horloge à événements récursifs distribués (AHER). Nous allons montrer que les AHER sont déterminables, donc fermés par complémentation, aussi que leurs problèmes respectifs d'inclusion de langue sont décidables (plus exactement, PSPACE-complet). De plus, nous proposons des extensions de la logique temporelle temporisée existante (EventClockTL) avec des horloges distribuées pour permettre la spécification de propriétés temporelles distribuées et temporisées. Cela nous donne la logique temporelle d'horloge d'événement distribuée (récursive) (LTHED) dont nous montrons qu'elle est PSPACE-complète pour les problèmes de satisfiabilité et de validité. Enfin, nous montrons l'applicabilité de AHER et LTHED sur un SDTR.

(ii) Horloges indépendantes multiples (HIM) : Un SDTR implique plusieurs processus en temps réel interconnectés (appelés ici composants) où chaque composant utilise ses propres horloges locales indépendantes fonctionnant à son propre rythme. Ainsi, un seul horodatage suffit pour les horloges globales, tandis que plusieurs horodatages prennent en charge des horloges locales indépendantes. Dans le chapitre 7, nous étendons la sémantique standard de AT et la bisimulation temporisée afin d'inclure la notion d'horloges multiples indépendantes.

Par conséquent, nous étendons la sémantique des systèmes de transitions étiquetées temporisées (STET) et ATHI [ABG$^+$08] pour travailler avec la notion de HIM et de mots multi-temporisés. Par conséquent, nous proposons un nouveau formalisme temps réel appelé Automates multi-temporisés (AMT) basé sur ATHI (et HIM). De plus, nous proposons une logique modale temporisée avec des horloges locales indépendantes, donnant la logique modale (multi-temporisée) ML_v. Nous étendons la théorie classique de la bisimulation temporisée [Cer93] avec la nouvelle notion de bisimulation multi-temporisée et AMT. Nous montrons que la bisimulation multi-temporisée est décidable (plus exactement, EXPTIME-complète). De plus, nous proposerons un algorithme efficace pour la bisimulation temporisée multi-temporisée en utilisant des techniques de raffinement. Enfin, nous montrons l'applicabilité de AMT et ML_v sur un SDTR.

(iii) Dérivées d'horloges distribuées (DHD) : Une DHD $\dot{x}$ est une dérivée temporelle de l'horloge x. Des comparaisons entre deux dérivées d'horloge ou une dérivée d'horloge avec une constante naturelle 1 peuvent être effectuées. Dans le chapitre 8, nous étendons la sémantique de AT afin d'inclure la notion de dérivées d'horloges distribuées. Par conséquent, nous étendons la sémantique AT et L_v [AD94] [LLW95] pour travailler avec la notion de DHD et multi-mots temporisés, donnant la logique modale multi-temporisée (dérivée) DML_v et les automates temporisés avec des dérivées d'horloge (ATDH). Nous montrons que la bisimulation multi-temporisée (dérivée) est décidable (plus exactement, EXPTIME-complète). De plus, nous proposerons un algorithme efficace pour la bisimulation multi-temporisée (dérivée) en utilisant des techniques de raffinement. Enfin, nous montrons l'applicabilité de ATDH et DML_v sur un SDTR.

Keywords : Vérification formelle, Automates temporisés, Logiques temporelles temporisées, Bisimulation temporisée.

ACKNOWLEDGEMENTS

The work described in this thesis was not, and could not have been, done in isolation. It involved the help and support of many people, to whom I owe a great debt of gratitude. First, I would like to thank my advisor, Pierre-Yves Schobbens, who believed that I could make a contribution after a difficult start. Thank you, Pierre-Yves, for your collaboration and suggestions over the past six years, and for introducing me to real-time temporal logic, model checking, and automata. Working with Pierre-Yves has been an excellent experience, especially for his expertise, understanding, and patience, which have greatly enriched mine with many helpful ideas and technical support.

I would like to express my gratitude to the University of Namur for providing funding for my Ph.D. thesis. I am also thankful to the FOCUS/NADI and PReCISE/-NADI research groups for their support and for allowing me to develop my thesis there.

I would also like to thank the jury members Prof. S. Akshay, Prof. Jean-Marie Jacquet, Prof. Jean-François Raskin, Prof. Stavros Tripakis and Prof. Wim Vanhoof for their careful reading of my thesis, their constructive comments and their suggestions and corrections. They helped me to improve my thesis and the presentation of the results. I would also like to thank the anonymous reviewers of the published and unpublished papers on which part of this thesis is based for their criticism and suggestions.

In particular, I would like to thank Dr. Gilles Perrouin, Prof. Paul Temple, Prof. Xavier Devroey and Dr. Moussa Amrani for their invaluable help throughout the process. I would also like to thank them for proofreading part of this thesis and for their helpful comments.

Special thanks to my colleagues in the PReCISE/NADI and FOCUS/NADI research groups, Sophie, Edilton, $Antoine^3$, Manel, Guillerme, Martin, Maxime, Loup, Tony, Pol, and all the other members of the Faculty of Computer Science. I am also grateful to the administrative staff of the Faculty of Computer Science, who have kindly helped me with the bureaucratic process all these years.

Finally, I would like to express my deepest gratitude to my $\heartsuit^3$, family and friends. I especially thank you, Emilie, for always covering my blind side, for understanding how much this thesis means to me, and for all the time we have spent together over the past ten years. I would like to thank my brothers, Robinson and Emerson, and my entire family, especially my mother, Miriam, and my father, Raul, for supporting me and encouraging me to follow my dreams.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

With the rapid growth of distributed computing and networking, the demand for large-scale and complex distributed applications has increased significantly over the past decade. Distributed Real-Time Applications are used to control and monitor Distributed Real-Time Systems (DRTS) such as aerospace systems, aircraft systems, robotics, nuclear power plants, and so on, and should maintain high assurance and quantitative real-time properties at all times. Many of the distributed real-time applications run on heterogeneous computer networks with numerous interconnected components or processes. Thus, there is a need for distributed applications that can operate within the time constraints and local clocks imposed on them. However, the risk of failures in distributed real-time applications is always present, which would result in numerous losses, including financial resources and human lives.

DRTS are structured into multiple communicating components (or processes) whose behavior depends on multiple timing constraints, and such components may be located on multiple computers distributed over a communication network. A DRTS can be classified as using: (1) synchronous clocks, if all of its components use the same global time, and (2) asynchronous clocks, if both of its components have their independent local clocks that are subject to clock drift [Cri96]. Synchronous and asynchronous clock models represent two ways of modeling and implementing

DRTS. However, the majority of current implementations of DRTS combine these two models, which is known as the timed asynchronous model. [Cri96]. In a timed asynchronous DRTS, each component has access to its local clock, which runs at the rate of local time [PSR94].

Formal verification methods and real-time formalisms have been used to verify the correctness of DRTS, but traditional real-time formalisms are not always adequate for reasoning about DRTS because they assume a unique, perfectly synchronous (Newtonian) measure of time. In real DRTS, such perfect clocks cannot be implemented. In contrast, using multiple clocks and associating them with multiple time stamps could be a compact and efficient representation. Therefore, it is crucial to remove the assumption of perfect clock synchronization by distributing clocks that can advance independently if they are in different components.

However, tools like UPPAAL [UPP], KRONOS [BDM$^+$98], TEMPO toolset [GMP13], IF toolkit [BGO$^+$04], HyTech [HHWT97] are all based on synchronous clocks. While some formalisms with local time, which characterize the concepts of local clocks of a DRTS, attract an increasing number of studies [BJLY98] [Kri99] [ABG$^+$08], little attention is dedicated to current literature to the semantics of multiple local time and the undecidability problem of these formalisms.

Hence, complex interactions between multiple local times and distributed behavior of DRTS must be rigorously studied. Lack of tools and formalisms that allow automatic verification of distributed real-time properties of DRTS emphasize the relevance of this thesis. This thesis strives for new formalisms for the specification and modeling of DRTS. Also, this proposal strives for developing computational tools which allow specifying, analyzing, and reasoning about the behavior of DRTS.

1.1 Context and problem statement

The most successful formalisms for modeling RTS and DRTS are Timed Automata (TA) [AD94], Event Clock Automata (ECA) [AFH94], and Recursive Event Clock Automata (RECA) [HRS98]. A TA is a finite automaton augmented with real-valued clocks. The standard semantics of TA is based on a Timed Labelled Transition System (TLTS), i.e. a run of a TA is given by a sequence of actions and timestamps. Also, TA assumes perfect clocks, where all clocks have infinite precision and are perfectly synchronized. This causes TA to have an undecidable language inclusion problem [AD94].

To restore the decidability of TA, Alur et al. [AFH94] proposed to restrict the behavior of the clocks. Thus, an event clock is reset when a given atomic proposition occurs. The values of the event clocks are deterministic, and thus ECA are determinizable, making the language inclusion decidable. However, the expressiveness of ECA is rather weak. Therefore, Henzinger et al. [HRS98] introduced the notion of recursive event. In a recursive event model (RECA), the reset of a clock is decided by a lower-level automaton (or formula). This automaton cannot read the resetting clock. Clock resets are thus still deterministic, but the concept of events is now much more expressive. Also, Henzinger et al. [HRS98] introduced the temporal logic of recursive event clocks (EventClockTL). EventClockTL has the same expressiveness as the Metric Interval Temporal Logic MITL (a decidable fragment of MTL where punctual constraints are forbidden) in interval semantics [AFH96]. However, the expressiveness of event clock models has still been criticized for specifying and verifying RTS and DRTS because, as with TA, ECA and RECA assume perfect clocks. All clocks have infinite precision and are perfectly synchronized.

In addition, there are other variants of TA called Distributed Timed Automata (DTA) and Timed Automata with Independent Clocks (icTA) are proposed by [Kri99] and [ABG$^+$08] to model DRTS, where the clocks are not necessarily synchronized. Constraints on the clocks are used to restrict the behaviors of the automata. DTA and icTA are neither determinizable nor complementable and their inclusion problems are undecidable [ABG$^+$08].

Extensions of timed modals and temporal logics, such as Timed Propositional Modal Logic (TPML), L_v [LLW95], and Timed Computation Tree Logic (TCTL [TY01]), have been used to specify (single-timed) real-time systems. However, in these logics, the information about independent clocks and distributed components observed in DRTS is modeled on a global setting [Ray15]. Consequently, these logics may not be suitable for explicitly specifying local timing properties that need to hold only in selected parts of the overall system. In essence, this means that the (single-timed) semantics of these logics is defined in terms of TLTS. However, there are logics that have been defined to capture aspects of distributed components and timing properties of DRTS: e.g., DRTL [MP90], APTL [WME93], among others. Roughly speaking, these logics allow the definition of formulas whose truth values depend on (or are relative to) only part of their underlying mathematical models. In the case of DRTL and APTL, these logics are an extension of Second-Order Logic (SOL) and First-Order Logic (FOL), where the set of formulas consists of constants, functions, predicates, universal and existential quantifiers, and logical connectives. In general, these logics are undecidable, but depending on which fragment is used, the resulting logic may be decidable.

One of the most commonly used methods for DRTS is model checking. This method automatically checks that all execution sequences of the system are a model of the formula representing the property (i.e., an exhaustive analysis of the state space). However, applying this method to DRTS quickly becomes intractable because the state space often grows exponentially with the number of components considered. One of the most effective and successful techniques currently available for reducing the state space is to merge states with the same behavior. For untimed systems, the notion of bisimulation [Mil89] is classically used for this purpose, and its natural extension for RTS, timed bisimulation, has been used to verify the preservation of sequential behavior and timed properties expressed in timed temporal logics (e.g., timed CTL [TY01] or L_v [LLW95]). Timed bisimulation has been shown to be decidable for TA [LLW95]. Model checking tools, such as UPPAAL [UPP], KRONOS [BDM$^+$98], TEMPO Toolset [GMP13], IF Toolset [BGO$^+$04], HyTech [HHWT97] and the analysis techniques [BDL$^+$06] [TY01] for RTS have been implemented under the sequential semantics of TA and TLTS. It can be concluded that the sequential semantics does not fully describe the behavior of DRTS because it does not consider interactions between processes with their associated local clocks running at different rates. In contrast, a distributed semantics for TA and Network of TA (NTA) has been introduced in [BC13], but the associated semantics remains in the classical setting of perfect clocks evolving at the same rate.

1.2 Related Work

There are several formalisms and tools for DRTS based on automata and TLTS. One of the most widely used formal frameworks for DRTS is TA, and several implementations and extensions have been considered. For example, Puri et al. [Pur98] studied the semantics of robustness TA where clocks can drift in a small bounded way, i.e., clocks can grow at independent rates in the interval $1 \pm \epsilon$ (for an arbitrary ϵ). Puri et al. [Pur98] showed that the reachability algorithm is incorrect when clocks drift, even by infinitesimally small amount, and subsequently proposed a region-based method for computing $Reach^{*}(S_{\epsilon})$, the set of reachable states for every drift (the limit as $\epsilon \to 0$), i.e., $Reach^{*}(S) = \bigcap_{\epsilon>0} Reach(S_{\epsilon})$. Wulf et al. [DWDMR04] proposed another perturbation model where the model is syntactically modified by relaxing the guards through a parametric increase of δ. Wulf et al. [DWDMR04] showed that the notion of robustness defined in [Pur98], and studied in other works [DK06], [Dim07], is closely related to the notion of implementability introduced in [DWDMR04], i.e., whether for some $\delta > 0$, the extended system model still satisfies the requirements expressed by the considered properties. Altisen et al., [AT05] studied whether a TA can be implemented on a given platform satisfying a desired property using its standard semantics and modeling, instead of extending it (i.e., given a TA $\mathscr{A}$, does there exist some $\delta > 0$ and δ clock drift implementation of $\mathscr{A}$, in which the properties of $\mathscr{A}$ are preserved). Altisen et al., [AT05] showed how to transform a TA into a program and how to model the execution of this program on a given platform as an assembly of TA. Swaminathan et al. [SFK08] considered a more realistic model of drifting clocks, where the clock resynchronization is taken into account. Sankur et al. [SBM14] studied the robustness problem in TA against guard shrinking. Sankur et al. [SBM14] provided a method for deciding whether shrinking all timing constraints (guards) of a TA, by possibly different amounts, results in a TA that preserves some time-abstract behavior and is not blocking. Sankur et al. [San15] studied the robustness analysis of real-time systems modeled by TA, where the goal was to compute a bound on the timing imprecisions so that the model satisfies a given specification. Sankur et al. [San15] proposed a semi-algorithm for infinitesimal analysis, which consists in finding a safe bound on imprecisions.

Krishnan et al. [Kri99] considered asynchronously distributed timed automata where clocks evolve independently in each component. Dima et al. [Dim07] proposed a distributed time asynchronous automata, where all components evolve independently of each other, the local times are incremented independently, which will increment the global time with the sum of the local increments. The timed languages defined by distributed time-asynchronous automata are strictly larger than the timed languages of TA. It is proved that this class is equivalent to the languages defined by a certain class of TA, called partitioned TA. Dima et al. [Dim07] showed that distributed time-asynchronous automata are more expressive than

TA. Akshay et al. [ABG$^+$08] focused on the untimed language of DTA. Ortiz et al. [OLS11] proposed a model that has the same expressiveness as Event Clock Automata (ECA) [AD94], but did not study possible bisimulation algorithms.

There are other formalisms, such as Timed Input/Output Automata (TIOA) [KLSV03], Hybrid Automata (HA) [Hen96], Hybrid Input/Output Automata (HIOA) [LSV03], Multi-Rate Timed Automata (MRTA) [HKPV98], Rectangular Hybrid Automata (RHA) [HKPV98] which are often used for modeling DRTS [KLSV10] [DLL$^+$10]. Consequently, TIOA, HA, MRTA, RHA and HIOA can be used to analyze Timed Distributed Algorithms (TDA), such as clock synchronization algorithms that use local clocks evolving at different rates [KLSV10]. TIOA, MRTA, RHA and HIOA can be viewed as nondeterministic state machines in which the internal state can change in two ways: (1) by an instantaneous discrete transition labeled by a discrete action, or (2) by a trajectory, which is a function describing the evolution of the state over a time interval [KLSV03]. HIOA are an extension of TIOA, where external variables model the continuous information flowing into and out of the system [KLSV10]. However, the reachability problem and simulation (and bisimulation) are undecidable for TIOA (and HA, RHA, MRTA and HIOA) [KLSV10], but decidable for TA and NTA.

A special class of HA are RHA [HKPV98]. Their characteristic property is that derivatives of continuous variables, also called flows, are chosen non-deterministically from a rectangular set, i.e. an interval, and resets, guards, and invariants are defined using rectangular sets. However, the reachability problem for RHA is undecidable [HKPV98]. A RHA is initialized because each continuous variable is reset at each transition that leading to a location with a different flow interval for the variable. Initialized Rectangular Hybrid Automata (IRHA) are the most expressive class of RHA for which unbounded reachability is decidable [HKPV98]. Singular hybrid automata (SHA) are a subclass of RHA [HKPV98]. Their continuous variables have constant rates, i.e. the flow function assigns a constant $c_i \in \mathbb{R}$ to each variable x_i at each location. The flow rate of a variable can be different at different locations. A SHA is then a RHA where for each flow function the upper and lower bounds are equal. The reachability problem for Initialized Singular Hybrid Automata (ISHA) is decidable [HKPV98]. Stopwatch Automata (SWA) are a subclass of RHA that only allow flow rates to be 0 or 1. Although the expressiveness gained by SWA is huge [CL00], this seemingly small variation makes the reachability problem undecidable [HKPV98]. Unlike TA, not every SWA is initialized. However, the reachability problem for non-initialized SWA is undecidable and initialized SWA (ISWA) can be polynomially encoded by TA. Multi-Rate Timed Automata (MRTA) is a subclass of SHA, where all continuous variables are so-called skewed clocks, i.e. they have the same flow rate in every location [HKPV98].

Thus, the decidability of reachability for IRHA works via transformation to a subclass of RHA [HKPV98]. (1) IRHA can be transformed into ISHA by replacing each continuous variable x by two continuous variables: x_l representing the lower bound and x_u the upper bound of x. (2)ISHA can be transformed into ISWA by setting for each variable the flows greater than 0 to 1 and scaling all invariants, guards, and resets concerning the corresponding variable accordingly. Finally, (3) ISWA can be transformed into TA, where the flow rate of each variable at each location is 1. The transformation is done by setting the flow rates of all continuous variables to 1 and adjusting the resets of these variables [HKPV98].

The notion of bisimulation for TA is studied in several papers [Cer93], [WL97], [TY01], [BCDL09], [BBLP06]. Cerans et al. [Cer93] gave a proof of decidability for timed bisimulation. Several techniques are used in the literature to provide algorithms capable of checking (bi-)simulation: Weise et al. [WL97] relied on a zone-based algorithm for weak bisimulation over TA, but no implementation is given. Bulychev et al. [BCDL09] studied timed simulation for simulation-checking games, for which an implementation is available from [BBLP06]. A region construction for timed bisimulation that is closer to our work was also considered by Akshay et al. [ABG$^+$08], but never implemented. Tripakis et al. proposed a time-abstract bisimulation over TA in [TY01]. Krishnan et al. [Kri99] and Ortiz et al. [OLS11] also manipulated clock drifts to manipulate DTA, but did not consider bisimulation.

In addition, TA, TIOA, and HIOA are supported by a variety of verification tools, such as UPPAAL [UPP], TEMPO Toolkit [GMP13], HyTech [ACH$^+$95] [HHWT97], Shrinktech [San13], KRONOS [BDM$^+$98]. UPPAAL [UPP] is a model checker for TA that performs forward reachability analysis with extrapolation. It can check the reachability properties of RTS with some extra features like bounded integer variables and broadcast channels. The TEMPO toolset is based on the semantics of TIOA [GMP13]. However, the reachability problem for TIOA and Linear Hybrid Automata (LHA) [ACH$^+$95] [HHWT97] is undecidable [HHWT97] [KLSV10]. HyTech [ACH$^+$95] [HHWT97] is a model checker for linear hybrid automata (LHA). Exact backward and forward computations can be performed, and thus reachability properties can be checked (but there is no guarantee that the computation will terminate). Many other operations can be performed on polyhedra, such as hiding variables (corresponding to projections), while loops, emptiness checks, etc. KRONOS [BDM$^+$98] is a model checker for TA that allows both exact and abstract backward and forward computations. A backward computation for TCTL [TY01] is also implemented in KRONOS. [BDM$^+$98]. Shrinktech [San13] is a tool that implements the simulation shrinkability algorithm presented in [SBM14]. The tool has been used to verify the shrinkability of several case studies, such as the Philips Audio Retransmission protocol, Fischer's Mutual Exclusion protocol (up to 4 agents), and some other asynchronous circuit models.

Bengtsson et al. [BGK$^+$02] presented a case study using the UPPAAL verification tool to verify an industrial audio control protocol with bus collision handling from Philips. Bengtsson et al. [BGK$^+$02] showed that the protocol behaves correctly when the error is bound to ±5% on all timing, and incorrectly when the error is bound to ±6%. Bengtsson et al. [BGK$^+$02] used UPPAAL to generate diagnostic traces, and they studied a buggy version of the protocol actually implemented by Philips in their audio products, and constructed a possible execution sequence that explains a known error.

There has been extensive research on extending modal logic to the setting of RTS. Similar to how DTA extends TA, modal logic has been extended to include quantitative timing information. Modal logic has been extended with time and recursion [LLW95]. Also in [LLW95], model checking and satisfiability problems (for a bounded number of clocks and values of constants) were shown to be decidable over TA. In [AL99], the model checking problem for the logic L_ν over TA is PSPACE-complete. In [LM14a], Recursive Weighted Logic (RWL) was studied, a modal logic that expresses qualitative and quantitative properties. The satisfiability problem for RWL is decidable by applying a variant of the region technique developed for TA. Several logics have been defined to capture aspects of quantitative timing information and distributed properties, such as DRTL [MP90] and APTL [WME93]. In these logics, it is possible to define formulas whose truth values depend on (or are relative to) only part of their underlying mathematical models. In the case of DRTL and APTL, these logics are an extension of Second-Order Logic (SOL) and First-Order Logic (FOL), where the set of formulas is composed of constants, functions, predicates, universal and existential quantifiers, and logical connectives from FOL. In general, this timed temporal logic does not use different action labels and delays, i.e. it is interpreted via Timed Labelled Transition System (TLTS).

1.3 Contributions

This thesis contributes several fully decidable formalisms that are effective to study the behavior and in particular the correct operation of RTS and DRTS:

- We remove the assumption of perfect clock synchronization. Here, inspired by [BJLY98, Kri99, ABG$^+$08, DL07, Dim03], we study the worst case: the clocks can advance independently if they are in different processes. However, Puri et al. [Pur98] and Wulf et al. [DWDMR04] studied the opposite case, where the difference between the clocks (drift) is infinitesimally small. While Akshay et al. [ABG$^+$08] studied only the untimed languages of their timed automata, namely the universal and existential languages, our first contribution is to define and study the corresponding timed languages. We extend the concept of timed languages to multi-timed languages for distributed timed specifications.
- We extend the Recursive Event Clock Automata (RECA) [HRS98] with distributed (a.k.a. independent) clocks, yielding the Distributed Recursive Event Clock Automata (DECA). We will show that DECA are determinizable, i.e. closed under complementation, and thus that their language inclusion problem is decidable (more precisely, PSPACE-complete). We also show the decidability and regularity of their existential and universal timed languages.
- We propose the (recursive) Distributed Event Clock Temporal Logic (DECTL), which is derived from EventClockTL. [HRS98] by controlling the use of independent clocks as follows: clocks have predefined associations with formulas of the logic, i.e. local clocks used in DECTL can be seen as an extension of the concept of event clocks used in Recursive Event Clock Automata (RECA). The event clocks were recursively linked to formula logic. We will show that the logic DECTL is decidable and, more precisely, PSPACE-complete.
- We extend the semantics of Timed Labelled Transition Systems (TLTS) and icTA [ABG$^+$08] to work with the notion of multi-timed. word and multi-timed semantics. Therefore, we will propose a formalism called Multi-timed Automata

(MTA) based on TA and icTA. Furthermore, we extend the classical theory of timed bisimulation [Cer93] with the new notion of multi-timed. bisimulation and MTA. We will show that multi-timed bisimulation is decidable (more precisely, EXPTIME-complete) and present two algorithms:
  (i) A forward reachability algorithm for the lockstep composition of two MTA, which will help us to reduce the state space exploration, and
  (ii) A decision algorithm for multi-timed bisimulation using the zone-based technique [BY04].
- We extend L_ν and Hennessy-Milner logics [HM85] [LLW95] with distributed (a.k.a. independent) local clocks, yielding the (multi-timed) modal logic ML_ν. We show that ML_ν is PSPACE-complete for the satisfiability and validity problem.
- We extend the TA semantics [ABG+08] to work with the notion of clock derivatives. Therefore, we will propose a formalism called Timed Automata with Clocks Derivatives (DMTA) based on TA and MTA. We will show that (derivative) multi-timed bisimulation is decidable (more precisely, EXPTIME-complete).
- We extend ML_ν and Hennessy-Milner logics [HM85] [LLW95] with distributed clock derivatives, yielding the timed modal logic DML_ν. We show that DML_ν is PSPACE-complete for the satisfiability and validity problem.
- We develop and implement a timed bisimulation tool (called MUTES) that allows to decide whether two DMTA (TA and MTA) are bisimilar.
- We develop and implement a model checking tool (called MIMETIC) that allows the specification and verification of distributed real-time properties.
- We develop and implement a tool called, MULTI-TEMPO, which allows modeling and simulation of DRTS with independent clocks.
- We show the applicability of DECA, DECTL, MTA, ML_ν, DMTA, and DML_ν to a DRTS.

1.4 Structure of the thesis

The remainder of this thesis is structured as follows:

(i) **Chapter 1** [**Introduction.**] In this chapter, we introduce the problems addressed in this thesis and give an overview of its structure.

(ii) **Part I**: **Background**
  - **Chapter 2** [**Preliminaries.**] In this chapter we introduce the mathematical concepts for the development of this thesis, such as automata, Transition Systems (TS), Timed Labeled Transition Systems (TLTS), Timed Automata (TA), Real-time Systems (RTS), and Distributed Real-time Systems (DRTS).
  - **Chapter 3** [**Modal and Temporal Formalisms.**] In this chapter, we recall the requirements and properties of formalisms that can define the temporal properties of distributed systems. These formalisms are LTL, CTL, and HML.

- **Chapter 4** [**Distributed and Real-Time Temporal Formalisms.**] In this chapter, we recall the two models we use in the thesis to represent real-time behavior: Timed Sequence and Interval Sequence. We also review the main formalisms that have been proposed to specify real-time properties: MTL, MITL, EventClockTL, L_ν, TA, ECA, RECA, and icTA.

(iii) **Part II**: **Contributions and Results**

- **Chapter 5** [**Distributed Event Clocks**] In this chapter, we define the formalisms Distributed Event Clock Temporal Logic (DECTL) and Distributed Event Clock Automata (DECA) in the context of timed interval sequences (continuous semantics). We also show the applicability of DECA and DECTL to DRTS.
- **Chapter 6** [**Multiple Independent Clocks**] In this chapter, we define the formalisms multi-timed automata (MTA) and (multi-timed) modal logic ML_ν in the context of multi-timed words. We show the applicability of MTA and ML_ν to DRTS.
- **Chapter 7** [**Distributed Clocks Derivatives**] In this chapter, we define the formalisms Multi-timed Automata with Derivative Clocks (DMTA) and Timed Modal Logic with Derivative Clocks DML_ν in the context of rate constraints. We show the applicability of DMTA and DML_ν to DRTS. We also show the design and implementation details of our multi-timed bisimulation algorithms for DMTA and our model checking tool.

(iv) **Part III**: **Discussion**

- **Chapter 9** [**Conclusions and Future Work.**] The conclusions and possible future work are summarized in Chapter 9.

1.5 Publications

The content of this thesis is based upon, reuses, and extends the following peer-reviewed publications of the author:

- **Proceedings of conferences**.
  - Ortiz Vega, J., Legay, A. and Schobbens, P-Y. Distributed Event Clock Automata. 16th International Conference on Implementation and Application of Automata (CIAA-2011), Blois, France, July 12-16, 2011. The contributions about Distributed Event Clock Automata will be included in Chapter 5.
  - Ortiz Vega, J., Legay, A. and Schobbens, P-Y. Distributed Event Clock Automata. 16th International Conference on Implementation and Application of Automata (CIAA-2011), Blois, France, July 12-16, 2011. The contributions about Distributed Event Clock Temporal Logic will be included in Chapter 6.
  - Ortiz Vega, J., Amrani, M. and Schobbens, P-Y. Multi-timed Bisimulation for Distributed Timed Automata. 9th International Symposium, NFM 2017 Moffett Field, Proceedings. Davies, M., Kahsai, T. and Barrett, C. (eds.) (NFM-2017), CA, USA, May 16-18, 2017. The contributions of Multi-timed Automata will be included in Chapter 5.

  - Ortiz Vega, J., Amrani, M. and Schobbens, P-Y. ML_ν: A distributed real-time modal logic. 11th International Symposium, NFM 2019, Proceedings. Badger, J. M. and Rozier, K. Y. (eds.). (NFM-2019), Houston, USA, May 7-9, 2019. The contributions about ML_ν will be included in Chapter 6.
- **Workshops**.
  - Jaime Cuartas, Jesus Aranda, Maxime Cordy, James Ortiz, Gilles Perrouin, Pierre-Yves Schobbens. MUPPAAL: Reducing and Removing Equivalent and Duplicate Mutants in UPPAAL, A-MOST workshop at ICST2023, Dublin, Ireland, 2023.
- **Technical Reports and Short Papers**.
  - James Jerson Ortiz Vega, Axel Legay, and Pierre Yves Schobbens. Distributed Event Clock Automata.
    http://www.info.fundp.ac.be/$\sim$jor/DECAReport/
- **Abstract and Newsletter**.
  - James Jerson Ortiz Vega, Axel Legay, and Pierre Yves Schobbens. Distributed Event Clock Automata.
    http://moves.vub.ac.be/$_$media/info/newsletter/newsletter$_$6.pdf
- **Seminars**.
  - InfoRum Seminar, Wednesday, 5 December, 2018, University of Namur, https://www.unamur.be/info,
  - MFV Seminar, Wednesday, 12 December, 2018, http://di.ulb.ac.be/verif/,
  - Grascomp Doctoral Day, Friday, 22 November, 2019,
    https://www.grascomp.be/events/gdd19/,
  - FOCUS seminar, Friday, 6 December, 2019,
    https://researchportal.unamur.be/activities/focus-research-seminar,
  - AVISPA seminar, 25 Years of AVISPA Research Group, Monday, 30 November, 2020, http://cic.javerianacali.edu.co/wiki/doku.php?id=grupos:avispa:25-years,
  - VeriDis seminar, Friday, 24 June, 2022
    https://team.inria.fr/veridis/seminar/.
  - Love seminar, Friday, 15 November, 2022
    https://lipn.univ-paris13.fr/an-alternative-multi-timed-semantics-for-modelling-the-behaviour-of-distributed-timed-systems/.
  - Hybrid Systems research seminar, Monday, 21 November, 2022
    https://ths.rwth-aachen.de/.
  - Systems & Control seminar, Thursday, 4 February, 2023
    https://www.maastrichtuniversity.nl/pieter.collins/extra-activities.
  - Formal System Analysis seminar, Thursday, 22 February, 2023
    https://www.tue.nl/en/research/researchers/jeroen-keiren.

Part I

Background

“We can’t solve problems by using the same kind of thinking we used when we created them.”

— Albert Einstein

CHAPTER 2

FORMAL METHODS FOR DISTRIBUTED AND REAL-TIME SYSTEMS

Distributed Systems (DS) consist of multiple autonomous components that communicate, collaborate, and interoperate over communication networks. However, these communication networks can be affected by some fundamental problems such as low bandwidth, high latency and instability. Therefore, it is difficult for a component to know the current status and timing of communication of other components. These limitations prevent DS from using a global time reference between all components. These limitations can be avoided if each component could always run independently. In addition, the time delay can affect the behavior of the components, and it can be one of the most challenging problems in developing correct DS.

Also, DS can interact with the physical world, which subjects them to certain real-time constraints. Systems with such real-time constraints are called Distributed Real-Time Systems (DRTS) or Distributed Real-Time Critical Systems (DRTCS).

Examples range from traffic light controllers to airplanes, from telecommunication networks to medical systems. In the last decades, several formal methods and real-time formalisms have been proposed to formalize and prove properties of DRTS. However, the traditional real-time formalisms for reasoning about Real-time Systems (RTS) are not always adequate for reasoning about DRTS. Therefore, it is crucial to develop computational models that allow describing, analyzing, and reasoning about the behavior of DRTS.

This chapter provides definitions and background information about the main concepts and terms used throughout this thesis. It starts with the basic concept of Distributed Systems (DS), including their important characteristics and their main components. Then, an overview of the basic principles of Real-Time Systems (RTS) and Distributed Real-Time Systems (DRTS) is presented. Then, a general overview of formal methods (FM) and model checking (MC) is presented. Section 2.1 introduces DS. Distributed and real-time systems are introduced in sections 2.2 and 2.3, formal methods, formal verification, and model checking are introduced in sections 2.4, and finally temporal, real-time, and distributed formalisms are introduced in sections 2.5, 2.6, and 2.7.

2.1 Distributed Systems

Distributed Systems (DS) consist of multiple components running on different computers connected by a network. Communication between components can be achieved by passing messages (i.e. FIFO channels or shared variables). In addition to communicating internally, components can also interact with the external environment or human operators. The external environment consists of physical devices such as sensors and actuators. Good examples of distributed systems include smart grids, telecommunications networks, electronic banking, and airline reservation systems.

The four main goals of a DS are [TS06]:

(i) Resource Sharing: An important goal of a DS is to make it easy for users to access remote resources and share them with others in a controlled manner.
(ii) Transparency: A transparent DS is a system that can hide the fact that its components and resources are physically distributed across many computers.
(iii) Openness: An open DS is a system that is easy to extend and has a common interface for easier interoperability.
(iv) Scalability: A DS should be scalable in terms of geography, administration, or size.

Over the past forty years, a variety of formalisms for specifying and verifying DS have been developed, such as the Labeled Transition System (LTS), Temporal Logics [Pnu77, MP92], Modal Logics [BdRV01], Dynamic Logics [Par84], Finite State Machines [Hie04], Petri Nets [Pet81], and others. They have been used as a framework for reasoning about behavioral properties of distributed systems, such as control flow, computed values, and ordering of events.

2.2 Real-Time Systems

Most DS are real-time systems (RTS). A RTS must obey strict requirements about the timing of its output actions. Thus, a RTS should ensure that it responds within strict time constraints (i.e. deadlines). Examples of RTS are engine control systems in vehicles, air traffic control, mobile devices, nuclear power plants, anti-lock braking systems, ticket-booking systems, ATMs, factory automation systems, flight control systems, etc. However, verifying the correctness of RTS is much more complex than for untimed systems. The correctness of RTS depends not only on its functionality, but also on the time constraints imposed. RTS interact with their external environment using stimulus and perception signals (see Figure 2.1).

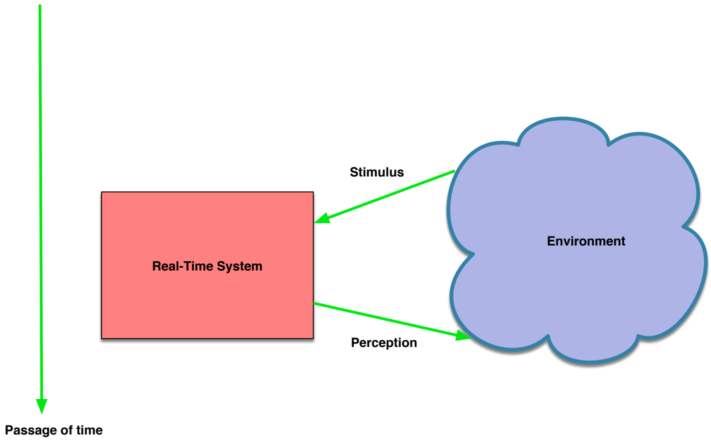


Figure 2.1: Real-Time Systems

RTS are classified by three types of deadlines:

(i) Hard RTS (HRTS). Failure to meet the deadline can result in catastrophic consequences, with possible loss of life,

(ii) Soft RTS (SHRT). The missed deadline is not critical, but it can degrade your quality of service,

(iii) Firm RTS (FRTS). A few missed deadlines may not cause a complete failure, but more than a few missed deadlines may cause a complete system failure.

A RTS can be modeled by combining hard, soft, and firm real time. Therefore, verifying the correctness of a RTS model by the classical approaches that rely on exploring the entire state space of the system is a challenging problem. Due to the impossibility to manually explore the state space of many systems, much research has focused on the use of formal methods for their verification (see Section 2.4).

For example, the behavior of RTS can be captured by Timed Labeled Transition System (TLTS) [Sch99, BLT94]. RTS can be specified in Timed Automata (TA) [AD94], and properties can be expressed in requirements languages such as Computation Tree Logic (CTL) [CE82], Timed Computation Tree Logic (TCTL) [ACD93], L_v [LLW95] or Timed Büchi Automata [Alu92]. Model-checkers, such as

TREX [ABS01], UPPAAL [UPP], KRONOS [BDM$^+$98] and LASH [LAS], can be used to explore the state-space of a system model, looking for counterexamples to the specified property.

2.3 Distributed Real-Time Systems

Distributed Real-Time Systems (DRTS) combine the characteristics of both DS and RTS, forming a distributed architecture where each component, called a node, is interconnected by a real-time communication network (see figure 2.2). However, DRTS are difficult to implement and understand because they are complex, dynamic, and completely variable. Their failure can have catastrophic consequences, so it is crucial to ensure their correctness. However, the correctness and performance of DRTS usually depend on their real-time properties as well as their behavioral ones. Therefore, there is a high demand for the implementation of a theoretical framework to specify and analyze the real-time properties of DRTS as well as their behavioral ones.

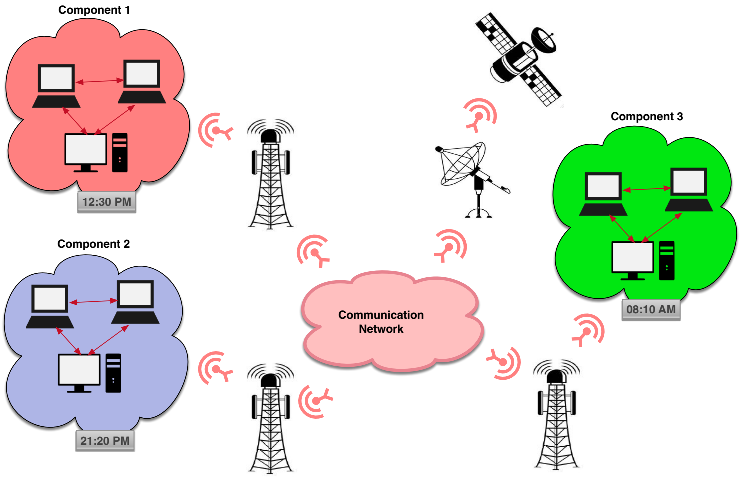


Figure 2.2: Distributed Systems

DRTS can be classified according to several criteria, the two most important of which are that [Cri96]:

(i) Type of communication[1]. It is helpful to differentiate between asynchronous and synchronous communications. The asynchronous communication consists of two primitives (blocking message send and blocking message receive) and synchronous communication consists of two primitives (non-blocking message send and blocking message receive).

[1] In this thesis, we use only point-to-point communications because other types of communications such as multicast and broadcast communications can be contemplated as point-to-point communications.

(ii) Type of time reference. It is useful to distinguish between same time and different time. At the same time means that clocks in different components are evolving at the same time (i.e., well-synchronized clocks). At different times means that clocks in different components evolve at different (independent) times (i.e., non-synchronized clocks).

Now, based on the two criteria presented above, the following formalisms for DRTS can be obtained:

(i) A formalism for specifying and analyzing DRTS with synchronous communication using clocks that evolve simultaneously. It is possible with this formalism to model DRTS where components are remotely distributed and communicate synchronously, but the mechanism of synchronized clocks is well-supported,

(ii) A formalism for specifying and analyzing DRTS with asynchronous communication using clocks that evolve simultaneously. It is possible with this formalism to model DRTS where components are remotely distributed and communicate asynchronously, but the mechanism of synchronized clocks is well-supported,

(iii) A formalism for specifying and analyzing DRTS with synchronous communication using clocks that evolve at different times. With this formalism, it is possible to model DRTS where components are remotely distributed and communicate synchronously. However, the mechanism of synchronized clocks is not supported.

The goal of this thesis is to establish formalisms for specifying and verifying properties of DRTS with independent clocks and communication delay.

2.4 Formal Methods

Defects in software development can cause all kinds of negative consequences and costs, from failures in the core system that can eventually lead to economic losses, to fatal cases that can cause loss of human life. Formal methods (FM) are used to identify these defects early in the software life cycle. The FM approach involves the use of various mathematical formalisms and tools that can be used to specify the behavior of a system and to formally verify the logical correctness of system requirements and properties with respect to the formalisms. FM has become a useful and automated technique used at an early stage during the software development process. Moreover, FM are essential for the development and verification of systems used for distributed safety-critical and mission-critical applications. Four approaches belong to the class of formal methods; they are not the only ones, but they are among the most important.

Tests Based on the Model. In this approach, the properties to be verified and the functional part of the system to be modeled and analyzed are described in a mathematical formalism. Second, the goal is to automatically generate a finite covering set of test cases (i.e., a set of tests that ensures that if the system passes these tests, then it satisfies the properties). Unfortunately, in many cases, such a covering set does not exist.

Static Analysis. This approach is used to statically analyze some properties of a code (i.e., without trying to compute all possible behaviors of the system). With this approach, it is possible to ensure that a variable is defined when it is used, or that there is no access to an array outside its domain if it is of constant size. However, dynamic properties cannot be tested with this approach.

Automatic Demonstration. This approach aims to construct a logical argument that demonstrates the correctness of the system with respect to a property. This correctness is expressed as a mathematical theorem in a test system. There are two different approaches to arguing the correctness of the system: automatic and semi-automatic. Assistants for automatic rule-based reasoning are then used to prove this theorem. However, a human operator must guide the assistants if they fail to prove some lemmas.

Model Checking. In the latter approach, the entire procedure is automatic and ensures that the system model satisfies a specification. This is represented in a logical formalism, and the correctness property is proved using algorithms.

All of the above approaches are complementary and are becoming increasingly prevalent in industrial environments [LDPM20]. Some well-known industrial examples are the development of the critical part of the Meteor metro at RATP and the critical parts of software developed by Airbus. The traditional testing methods are also important because they allow to study a model closer to the real system. However, this is not always possible, for example when using the real system is too expensive or too dangerous.

Since our work is in the context of automatic model verification, we will present this approach in more detail below.

2.4.1 Model Checking

Model checking involves three main steps, which are shown in Figure 2.3.

(i) **System Modeling.** The system under consideration can be given in the form of a real system (physical system or software code) or as a description of its behavior (e.g., a protocol). This system is translated into a mathematical formalism $\mathscr{M}$.

(ii) **Specification Modeling.** The model under consideration is often given by a specification. This model is also translated into a mathematical formalism, usually a logic, and gives a specification formula φ.

(iii) **Verification.** The model checking algorithms are then used to determine whether the system model satisfies the formula expressing the specification, which is noted $\mathscr{M} \models \varphi$. If it does not, the algorithm will highlight inconsistencies in the descriptions of the system and its properties. Otherwise, we get a guarantee that the model $\mathscr{M}$ satisfies the property expressed by φ.

The critical factor in model verification is the hardness of the problems studied. Thus, for the classes of too general models or properties, there is no verification of algorithms. And even for simple properties, scaling is often difficult, since the size of the studied systems can be very large. The complexity of verification algorithms depends on both the class of model and the logic specification under consideration.

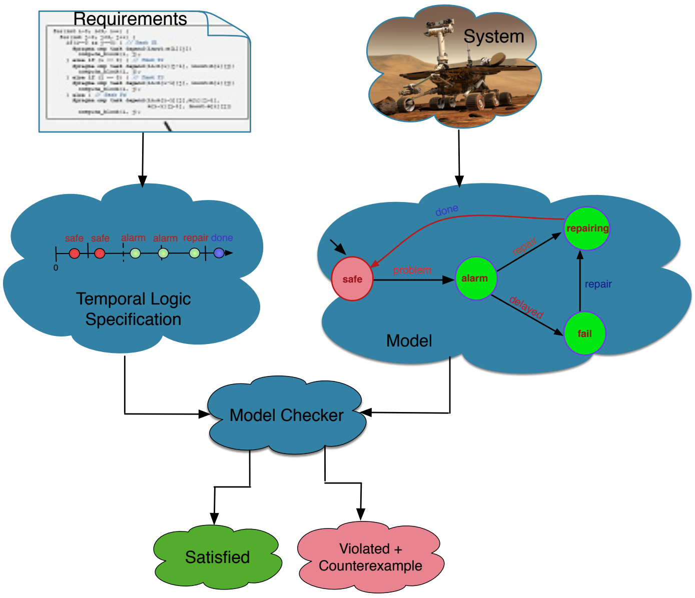


Figure 2.3: Model Checking

On the other hand, a weakness of formal methods comes from the difference that can exist between models and real system analysis. In fact, the modeling phase, which is to dive into a mathematical analysis of the system, although necessary, often simplifies the system. It is therefore natural to focus on the properties of conservation when the model is real, in order to obtain a model closer to a real system. The development of techniques to ensure the implementability of the models, i.e. to allow the transfer of the properties shown for the models to the systems themselves, is a key challenge. However, the basic theories for DRTS reasoning are not yet fully established, and some important questions remain open.

2.5 Temporal Aspects

Several formalisms have been proposed to model the temporal aspects of DS. Modal and temporal logics are the formalisms used to specify most of the behavioral properties of DS. Modal logics are formalisms that extend classical logic (first-order (FO) and higher-order logic) by adding modal operators that qualify truth. Modal logics can be extended in very simple ways, which can turn out to be extremely expressive. They can be used to express temporal properties by extending them with fixpoint operators. Modal μ calculus (L_μ) [Koz83] is an expressive temporal logic with modalities to reason about the actions that can be performed in a DS.

Temporal logics are special variants of modal logics, with modalities for reasoning about how the truth of a proposition can change over time. Temporal logics have at least one operator to perform arbitrarily many sequences of steps. Therefore, temporal logics can be used to specify properties of the behavior of a system in time. Temporal logics can be divided into two types of semantics. linear time [dBdRR89, BAMP81, CE82] and branching time [EH86, Sti87] semantics. In a linear time semantics, a DS can be viewed as an infinite sequence of states and events (i.e., traces), where the events are plotted on the states of the modeled DS. Traces can be used to describe infinite paths through labeled transition systems (LTS). In a branching time semantics, a DS can be viewed as an infinite tree of states, where each branch of this tree represents an execution (i.e., a trace) of the modeled DS, and each state may have several successor states. A branch describes an (infinite) tree by a Computational Temporal Logic (CTL).

Other formalisms have been proposed to model the temporal aspects of DS, such as monadic theories (First Order Monadic Logic (FOML), Second Order Monadic Logic (SOML)), automata (Büchi automata (BA) [Büc62]). In several papers, [Büc62, GPSS80, MW84], it is possible to find various proofs of the expressive equivalence between modal logics, classical logics, and finite automata.

The decidability of these formalisms and in particular their closure under boolean operations (union, intersection, complement) and their satisfiability (emptiness) are studied. The above formalisms are decidable for some properties if there is a finite procedure to prove or disprove these properties. The formalisms LTL, BA, QTL, FOML, SOML are fully decidable. There are several tools for verifying DS, see for example SPIN. [DGLM99, HPV00], MOCHA [AHM$^+$98], PROMELA [GMP04] and LASH [LAS].

2.6 Real-Time Aspects

As we noted earlier, temporal formalisms could be used to express temporal properties of a behavioral DS. Again, such properties are qualitative and can be described as a possibly infinite set of traces. However, to specify and reason about real-time temporal properties of RTS, quantitative timing information about the duration of events must be added to traces. Real-time temporal formalisms exist in many variants. Some are extensions of LTL, Finite State Automata (FSA), LTS, BA, and CTL. There are two alternatives for modeling quantitative time information: discrete time semantics, where the time values are non-negative integers, and dense time semantics, where the time values are real numbers.

There are several proposals for quantifying time information (discrete time semantics and dense time semantics) in temporal formalisms, such as adding bounds to existing operators (Metric Temporal Logic (MTL) [AH93]) and adding an interval to existing operators (Metric Interval Temporal Logic (MITL) [AFH96]) of temporal logic LTL, or the introduction of clocks and constraints on those clocks (Clock Temporal Logic ClockTL) [AFH96], Timed Propositional Temporal Logic (TPTL) [AFH96], Timed Modal Logic (L_v) [LLW95], Event EventClockTL [HRS98], and Timed Automata (TA) [AD94].

The complexity of existing real-time temporal formalisms depends on the (discrete time and dense time) semantics and the operators used. Many combinations lead to undecidable formalisms [AH93] [AFH96] [AH91]. In a dense time semantics, for example, the ability to express properties such as every event p is followed by an event q after exactly 1 unit of time leads to undecidability [AFH96]. However, MITL is a decidable formalism with temporal constraints and interval on temporal operators. Its complexity is EXPSPACE. [AFH96].

TA [AD94] are one of the most successful formalisms for modeling RTS. A TA is a finite state machine augmented with real-valued clocks. The model of TA assumes perfect clocks: all clocks have infinite precision and are perfectly synchronized. The theory of TA allows solving certain verification problems for RTS, such as reachability and safety problems. However, language inclusion is undecidable for TA. [AD94]. This is because TA is not determinable. Thus, one must either work with deterministic specifications or with a restricted class of TA that has the necessary closure properties.

The ECA [AFH94], are a subclass of TA that allow non-determinism and are closed under boolean operations, including complementation. The key feature of these automata is that they have a pair of event clocks (EC x_p, y_p) associated with each atomic proposition p. The clocks record the time that has elapsed since the last occurrence of the associated atomic proposition p, as well as the time that will elapse before the next occurrence of the atomic proposition p.

However, the expressiveness of ECA is rather weak. Furthermore, this logic violates the substitution principle: Every proposition should be replaceable by a formula. Therefore, [HRS98] introduced the notion of a "recursive" event. In a recursive event model, the reset of a clock is decided by a lower-level automaton (or formula). This automaton cannot read the clock it is resetting. Clock resets are thus still deterministic, but the concept of "event" is now much more expressive. The temporal logic of recursive event clocks (variously called SCL [RS97] or Event-ClockTL [HRS98]) has the same expressiveness as Metric Interval Temporal Logic MITL [AFH96].

2.7 Distributed Real-Time Aspects

DRTS often operates under strict timing constraints and shares resources among multiple distributed components. However, both distributed components and timing constraints make the design of a DRTS challenging. Another important characteristic of DRTS is their unpredictable (non-deterministic) nature, because such DRTS are often asynchronous, and it is difficult to accurately estimate the communication delay between any components. Thus, delay and asynchronous communication can cause serious difficulties in the design and development of DRTS. To create correct DRTS, it is necessary to analyze the effects of delay and asynchrony communication on the behavioral and real-time properties of the DRTS. Formal methods, such as model checking, have been used to verify the correctness of DRTS. Model checking of DRTS quickly becomes intractable because the state space often grows exponentially with the number of components considered. One of the most effective and successful techniques currently available for reducing the state space is to merge states with the same behavior. For untimed systems, the notion of bisimulation [Mil89] (see section 3.5) is classically used for this purpose, and its natural extension for RTS, timed bisimulation, has been used to verify the preservation of sequential behavior and real-time properties expressed in timed temporal logics (e.g., Timed CTL [TY01] or L_ν [[LLW95]).

Several real-time formalisms have been proposed to formalize and prove properties of DRTS. However, real-time formalisms are not always adequate for reasoning about DRTS. Therefore, it is crucial to develop computational models that allow describing, analyzing, and reasoning about the behavior of DRTS. Networks of TA and Timed Petri Nets (TPN) [AD94] [Mer74] are traditional real-time formalisms for modeling DRTS and have become popular as a modeling language for several model checkers such as UPPAAL and LASH. [UPP] [LAS].

There are other formalisms for modeling DRTS such as Distributed Timed Automata (DTA) [Kri99, ABG+08]. A DTA consists of several local TAs, called processes. Each process has clocks. The clocks of the same process evolve synchronously, but independently of the clocks of the other processes. In [ABG+08], DTA are not studied much. Instead, their product is computed first, resulting in the class of TA with independent clocks (icTA). icTA were described in [ABG+08] and are neither determinizable nor complementable. Their emptiness problem can be solved using the region construction [ABG+08], but their universal and inclusion problems are undecidable.

2.8 Wrap up

In this chapter, we introduced some basic concepts and formal methods used for the specification and verification of RTS and DRTS. We presented the basics of distributed and RTS and then gave an introduction to RTS, DRTS, formal methods, formal verification, model checking, and the models used to represent distributed and RTS.

CHAPTER 3

PRELIMINARIES

This chapter provides an introduction to the field of mathematical concepts, languages and untimed systems, temporal logics, timed systems, and the theory of computational complexity. It also introduces the notation and terminology used throughout the thesis. Readers may also consult additional information in standard textbooks, such as [Win93], [VvdPGS97] for mathematical concepts, [HMU06, BK08, Tho02] for automata, [Eme90, MP95, BK08] for temporal logics and model checking, [Mah05, DW07, Doy06, Alu92] for timed systems, [Pap94, CH97] for computational complexity. The relation between automata and logics is given in [Tho97].

This chapter is structured as follows. In section 3.1, we show some general mathematical notations that we will use throughout the thesis. In section 3.2, we present some general concepts about regular languages and words. We review necessary preliminaries from transition systems in section 3.3. We review necessary preliminaries from automata theory in section 3.4. Bisimulation and equivalence theories are reviewed in section 3.5. Finally, in section 3.6 we review standard definitions and results from computability and complexity theory.

3.1 Mathematical Concepts

We will first go through some mathematical concepts and introduce the notations we use for them in this thesis.

3.1.1 Sets

In mathematics, a (finite or infinite) set X is a collection of objects from a known universe Ω. The notation $x \in X$ ($x \notin X$) is used to denote that the object x is (not) an object of X. The finite set without an object is denoted as $\emptyset$ and the number of objects of a set X is denoted as $|X|$. A set can be specified by enumerating its objects, such as $\{0,1\}$ or $\{a,b,c,\cdots\}$. A set can also be specified by a property $\mathbb{P}$. The set $\mathbb{P} = \{p \mid p \in \mathbb{P}\}$ is the set of all properties of $\mathbb{P}$. For example, the set of odd natural numbers is denoted by $\{n \in \mathbb{N} \mid n\ mod\ 2 = 1\}$.

A subset Y of X is denoted by $X \subseteq Y$, where every object of X is also an object of Y. The notation $X \nsubseteq Y$ denotes that X is not a subset of Y. The set containing the exact subsets of a set X is called the power set of $2^X = \{Y \mid Y \subseteq X\}$. The union of the sets X and Y is denoted as $X \cup Y = \{x \mid x \in X\ or\ x \in Y\}$ and their intersection as $X \cap Y = \{x \mid x \in X\ and\ x \in Y\}$. The difference between the sets X and Y is the set of all objects of X that are not objects of $X \backslash Y = \{x \in X \mid x \notin Y\}$. The set $\mathbb{N} = \{0,1,2,3,\cdots\}$ denotes the set of natural numbers, the set $\mathbb{R}$ denotes the set of reals, $\mathbb{R}_{\geq 0}$ denotes the set of non-negative reals, and the set $\mathbb{Q}$ denotes the set of all rational numbers ($\mathbb{Q}_{\geq 0}$, the set of non-negative rational numbers). For any $x \in \mathbb{R}$, $fract(x)$ denotes the fractional part of x, and $\lfloor x \rfloor$ denotes the integral part of x, i.e., $x = \lfloor x \rfloor + fract(x)$.

In this thesis, $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ sets are often used. The set of sets is denoted as $\bigcup X = \{a \mid a \in x\ for\ some\ x \in X\}$, the intersection as $\bigcap X = \{a \mid a \in x\ for\ all\ x \in X\}$, and the product of X and Y is denoted as $X \times Y = \{(x,y) \mid x \in X\ and\ y \in Y\}$ (the set of all ordered pairs of objects from X and Y, respectively). In general, one can define the set of ordered n-tuples $(x_1, x_2, ..., x_n)$ from a product of n sets $X_1 \times X_2 \times \cdots \times X_n$.

3.1.2 Relations and Functions

A binary relation $\mathscr{R}$ between two sets X and Y is a subset of the Cartesian product $X \times Y$. If an element $x \in X$ is related by $\mathscr{R}$ to an element $y \in Y$, we often denote this fact by writing $x\mathscr{R}y$ instead of $(x,y) \in \mathscr{R}$. If $\mathscr{R} \subseteq X \times X$, we say that $\mathscr{R}$ is a binary relation on X. If $\mathscr{R}$ is a binary relation between X and Y, and $\mathscr{S}$ is a binary relation between Y and Z, then $\mathscr{R}\mathscr{S}$ is a binary relation between X and Z such that $xRSz$ if and only if there is some $y \in Y$ such that xRy and ySz.

A function f from the set X to the set Y is a binary relation from X to Y such that for every $x \in X$ there is at most one $y \in Y$ such that $(x,y) \in f$. If there is such a y, then $f(x)$ is said to be defined, and we write $y = f(x)$. If there is no such y, then we say that $f(x)$ is undefined. To denote that f is a partial function from X to Y, we write $f : X \hookrightarrow Y$. A partial function f from X to Y is called a total function if it is defined for each $x \in X$. To express that f is a total function from X to Y, we write $f : X \rightarrow Y$.

3.1.3 Asymptotic Notations

In this thesis, we use the following standard asymptotic notations, especially when measuring the computational complexity of a problem. We are only interested in the asymptotic behavior of its resource consumption. Our goal is to compare problems and see if one problem is much harder than another. For this purpose, we introduce the "big-oH" notation. Let f and g be functions from $\mathbb{N}$ to $\mathbb{N}$. Then $f(n) = O(g(n))$ if there exists real c and positive integer n_0 such that, for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$. Note that the statement $f(n) = O(g(n))$ means that $f(n)$ and $g(n)$ asymptotically have the same growth rate, or that they differ only by some linear factor. We write $f(n) = \omega(g(n))$ if the opposite is true, i.e. $g(n) = O(f(n))$. If $f(n) = O(g(n))$ and $f(n) = \omega(g(n))$, we write $f(n) = \theta(g(n))$.

3.2 Regular Languages

An alphabet Σ is a finite set of characters or symbols. A word is a finite sequence of Σ symbols. The empty word is denoted by ϵ. The set of all words above Σ is Σ^*. The star operation (Kleene) is a unary operation defined as $\Sigma^* = \bigcup_{i \in \mathbb{N}} \Sigma^i$ where $\Sigma^i = \{\sigma_1 \ldots \sigma_i \mid \forall j : 1 \leq j \leq i : \sigma_j \in \Sigma\}$ for any $i \in \mathbb{N}$ denotes the set of all sequences of length i of elements of Σ. The set Σ^+ is $\Sigma^* \backslash \{\epsilon\}$. A language $\mathscr{L}$ is a subset of words, i.e. $\mathscr{L} \subset \Sigma^*$, then if $\mathscr{L}$ is finite, we say that it is a finite language. Basic operations on languages include

- Union languages: $\mathscr{L}_1 \cup \mathscr{L}_2 = \{\sigma \mid \sigma \in \mathscr{L}_1 \ or \ \sigma \in \mathscr{L}_2\}$,
- Intersection languages: $\mathscr{L}_1 \cap \mathscr{L}_2 = \{\sigma \mid \sigma \in \mathscr{L}_1 \ and \ \sigma \in \mathscr{L}_2\}$,
- Concatenation languages: $\mathscr{L}_1 \cdot \mathscr{L}_2 = \{\sigma_1 \cdot \sigma_2 \mid \sigma_i \in \mathscr{L}_i, i = 1,2\}$,
- Closure languages (Kleene-star): $\mathscr{L}^* = \bigcup_{i \in \mathbb{N}} \{\sigma_1, \cdots \sigma_i \mid \forall j : 1 \leq j \leq i : \sigma_j \in \mathscr{L}\}$.

Words of Σ^* can be ordered using the prefix relation. A word u is a prefix of a word v, and write it $u \sqsubseteq v$ if there is a word w such that appending w to u yields v (i.e., $\exists w \in \Sigma^* : uw = v$). Also, u is a strict prefix of v if there is a w such that $uw = v$ and $w \neq \epsilon$.

An infinite sequence (ω) of symbols in Σ is represented by an ordered tuple of an infinite number of elements. The set of all such infinite sequences is called $\Sigma^{\leq \omega}$. Such a sequence is called a ω word. A set of such ω-words is called a ω-language.

3.3 Untimed Labelled Transition Systems

An abstract (mathematical) formalism can be used to (automatically) analyze the behavior of a system over time. The behavior of a system can then be described as an entity with states and transitions related to states. The transitions can be autonomous or stimulated by the environment. It is important to see that the behavior of the system is represented during such a state change. An abstract (mathematical) formalism that captures the behavior of untimed systems is called a Labelled Transition System (LTS). Propositional LTS is a formalism where it is possible to associate the propositions used in the specification of a system with states of the modeled system. Let $\mathbb{P}$ be a finite set of (propositional) atoms. Each proposition describes some property of the modeled system that may (or may not) hold in any of its states. Let $2^{\mathbb{P}}$ be the set of all subsets of $\mathbb{P}$, the labeling function $\gamma : Q \rightarrow 2^{\mathbb{P}}$ maps each state in Q (i.e., states in the modeled system) to the subset of propositions that hold in $2^{\mathbb{P}}$. A character (or symbol) is an element of a finite set Σ. In chapter 6 we will use the definition of a symbol as a propositional valuation over $\mathbb{P}$, so we set $\Sigma = 2^{\mathbb{P}}$.

Definition 1. *A LTS is a tuple $\mathscr{A} = (Q, q_0, \Sigma, \rightarrow_{lts}, \gamma)$ such that:*

(i) Q is a finite set of states,

(ii) $q_0 \in Q$ is the initial state,

(iii) Σ is a finite alphabet,

(iv) $\rightarrow_{lts} \subseteq Q \times \Sigma \times Q$ is a finite set of transitions,

(v) $\gamma : Q \rightarrow \Sigma$ is a function which labels each state $q \in Q$ with the set of atomic propositions (Σ) are true in that state.

The transitions from state to state of a LTS are noted as follows: (q, a, q') is denoted by $q \xrightarrow{a} q'$, if $a \in \Sigma$ and $(q, a, q') \in \rightarrow_{lts}$. A proposition $p \in \mathbb{P}$ is (true) in state $q \in Q$, denoted as $q \models p$, since s assigns a truth value true to p, otherwise $q \models \neg p$ assigns false. In chapter 6 we will use the notation $\Sigma = 2^{\mathbb{P}}$.

3.3.1 Trace Semantics

In a modeled DS with finite states, each state can be characterized by a finite set of propositions ($p \subseteq \mathbb{P}$) or a finite alphabet ($a \subseteq \Sigma$). Now we formally define traces.

Definition 2. *A trace σ over the alphabet Σ is an infinite sequence $\sigma = \sigma_0\sigma_1\ldots$, such that for every $i \geq 0$, $\sigma_i \in \Sigma^{\leq\omega}$. Note that an infinite trace can be seen equivalently as a function from $\mathbb{N}$ to Σ.*

In the following we use σ_i to denote the i-th letter of the sequence σ, this i-th letter is often called the i-th state of σ. We often use traces over the alphabet $\sigma = 2^{\mathbb{P}}$, i.e. the elements of the alphabet are the subsets of the set of propositions $\mathbb{P}$.

3.4 Finite Automata

A Finite Automaton (FA) is an abstract mathematical model of a system with input symbols, states, and a set of transitions from state to state that occur on input symbols from the alphabet Σ. FA is another formalism for representing regular languages. The FA has five elements or tuples.

Definition 3. *A (non-deterministic) Finite Automaton (NFA) is a tuple $\mathscr{A} = (Q, q_0, \Sigma, \rightarrow_{NFA}, Q_F)$, such that :*

(i) Q is the set of states,

(ii) $q_0 \in Q$ is the initial state,

(iii) Σ is a finite alphabet,

(iv) $\rightarrow_{NFA} \subseteq Q \times \Sigma \times Q$ is the transition relation,

(v) $Q_F \subseteq Q$ is a set of accepting (or final) states.

A NFA can be represented as in Figure 3.1. The states are rounded boxes and arrows represent transitions between states. Final states are denoted by a double line (right-most state), while initial states have a dangling incoming arrow (left-most state). The graph of Figure 3.1 represents the automaton $\mathscr{A}$ with:

(i) $Q = \{q_0, q_1\}$,
(ii) $q_0 \in Q$,
(iii) $\Sigma = \{a, b\}$,
(iv) $\rightarrow_{NFA} = \{(q_0, a, q_0), (q_0, b, q_0), (q_0, a, q_1), (q_1, b, q_1)\}$,
(v) $Q_F = \{q_1\}$.

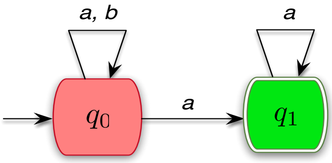


Figure 3.1: A NFA $\mathscr{A}$

For such a NFA $\mathscr{A}$, a sequence of states $q_0 q_1 \dots q_n \in Q$ is called a run on a word $a_1 a_2 \dots a_n \in \Sigma^*$ iff for every $0 < i \leq n$, there is a transition $(q_{i-1}, a_i, q_i) \in \rightarrow_{NFA}$ (Hence, often the notion of transition relation used is: $q_{i-1} \xrightarrow{a_i} q_i$). It is an initial run if $q_0 \in Q$ and it is final if $q_n \in Q_F$. We say that an automaton accepts a word u if there is some run $q_0 q_1 \dots q_n$ such that:

- $q_0 \in Q$, and
- $q_n \in Q_F$.

The language of a FA $\mathscr{A}$ is the set of all words accepted by this automaton: $\mathscr{L}(\mathscr{A}) = \{w \in \Sigma^* \mid$ there exists an accepting run for w in $\mathscr{A}\}$. Alternatively, we say that $\mathscr{A}$ recognizes $\mathscr{L}(\mathscr{A})$. A language is regular if there is a FA that recognizes it. Thus, languages recognized by FA are closed under Boolean operations (union, intersection, complement), difference, concatenation, and Kleene-star. As a convention, we will use W^c to denote the complement of W, i.e. $\sigma^* \setminus W$.

A FA is deterministic (DFA) if there is at most one outgoing transition labeled by every letter, from every state:

$$\forall q \in Q : \forall a \in \Sigma : |\{q' \mid \rightarrow_{DFA} (q, a, q')\}| \leq 1.$$

It is complete if additionally there is at least one outgoing transition labeled by each letter from every state:

$$\forall q \in Q : \forall a \in \Sigma : |\{q' \mid \rightarrow_{DFA} (q, a, q')\}| = 1.$$

A FA is not-deterministic (NFA) if there is at least more than one outgoing transition labeled by every letter, from every state:

$$\forall q \in Q : \forall a \in \Sigma : |\{q' \mid \rightarrow_{DFA} (q, a, q')\}| \geq 1.$$

The NFA can be determinized, i.e. transformed into a DFA that accepts the same language. This operation can cause the number of states to grow exponentially. So every language $W \subseteq \Sigma^*$ is regular if it is accepted by some NFA if it is accepted by some DFA.

3.5 Bisimulation and Equivalence

DS can be modeled by a set of interacting modules. Each of these modules can be thought of as a set of distinct components that can be designed, specified, and verified independently. The compositions of these components can be modeled using a set of operations such as sequential, parallel, or concurrent composition to build larger and more complex compositional systems. They quickly become intractable because their size grows exponentially with the number of components considered. This is due to combinatorial explosions that can be caused by an exponential increase in the number of state spaces (i.e., the state explosion problem) of the system model, generated by a linear increase in the number of running processes in a system under verification. Recent work in formal methods has proposed new tools and techniques to facilitate the exploration and reduction of state explosions. One of the most effective and successful techniques currently available for state explosion exploration and reduction is the generation of equivalent state spaces with respect to some desired properties of the system. A well-known technique for generating equivalent state spaces is the partition refinement technique (e.g., an application is the minimization algorithm [PT87]). The partition refinement technique is based on an iterative procedure in which a partition is refined by splitting its state space to obtain an equivalent state space of much smaller size with respect to some properties. Thus, several states in the state space are equivalent, and states belonging to equivalent states cannot be distinguished by semantic equivalences [HM85, Mil89]. Therefore, smaller state spaces are obtained in which verification and analysis of the properties of the original systems can be feasible. To reason about behavioral equivalence between different states (or components) of a system, the notion of bisimulation [HM85, Mil89] is used. The idea behind bisimulation equivalence is to consider two states (or components P, Q) and an equivalence relation, denoted by $\cong$, and then consider whether the two states are equivalent with respect to the equivalence relation (e.g., $q_1 \cong q_2$ or $P \cong Q$). Bisimulation equivalence is defined on the states of a given LTS or between different components of a DS. Bisimulation equivalence with observable actions of a LTS is called strong bisimulation. Bisimulation equivalence with abstractions of internal actions is called weak bisimulation. In this thesis we will use strong bisimulation.

3.5.1 Bisimulation

Let $\mathscr{H}_1$ and $\mathscr{H}_2$ be two LTS over the set of actions Σ. Let $Q_{\mathscr{H}_1}$ (resp., $Q_{\mathscr{H}_2}$) be the set of states of $\mathscr{H}_1$ (resp., $\mathscr{H}_2$). Let $\mathscr{R}$ be a binary relation over $Q_{\mathscr{H}_1} \times Q_{\mathscr{H}_2}$. We say that $\mathscr{R}$ is a strong timed bisimulation whenever the following transfer property holds:

Definition 4. *[Cer93] A strong bisimulation over LTS $\mathscr{H}_1$, $\mathscr{H}_2$ is a binary relation $\mathscr{R} \subseteq Q_{\mathscr{H}_1} \times Q_{\mathscr{H}_2}$ such that, for all $q_{\mathscr{H}_1} \mathscr{R} q_{\mathscr{H}_2}$, the following holds:*

(i) For every transition $q_{\mathcal{H}_1} \xrightarrow{a}_{\mathcal{H}_1} q'_{\mathcal{H}_1}$ with $a \in \Sigma$, there exists a matching transition $q_{\mathcal{H}_2} \xrightarrow{a}_{\mathcal{H}_2} q'_{\mathcal{H}_2}$ such that $q'_{\mathcal{H}_1} \mathcal{R} q'_{\mathcal{H}_2}$ and symmetrically.

Two states $q_{\mathcal{H}_1}$ and $q_{\mathcal{H}_2}$ are bisimilar, written $q_{\mathcal{H}_1} \cong q_{\mathcal{H}_2}$, iff there is a bisimulation that relates them. $\mathcal{H}_1$ and $\mathcal{H}_2$ are bisimilar, written $\mathcal{H}_1 \cong \mathcal{H}_2$, if there exists a bisimulation relation $\mathcal{R}$ over $\mathcal{H}_1$ and $\mathcal{H}_2$ containing the pair of initial states.

3.6 Computability and Complexity

In this section, we will recall some standard concepts from computability and complexity theory [Pap94] [CH97].

3.6.1 Turing Machine

Here, we will briefly recall the definition of Turing Machine (TM).

Definition 5. *A (nondeterministic) TM (NTM) is a tuple $\mathcal{M} = (Q, q_0, \Sigma, \Sigma_\Delta, \rightarrow_{TM}, Q_F)$, such that :*

(i) Q is the set of states,
(ii) $q_0 \in Q$ is an initial state,
(iii) Σ is a finite alphabet,
(iv) $\Sigma_\Delta \cup \$$ is a finite stack alphabet, where Δ is the blank symbol,
(v) $\rightarrow_{TM}$: $(Q \times \Sigma_\Delta) \times (Q \times \Sigma_\Delta \times \{L, R\})$ is a transition function,
(vi) $Q_F \subseteq Q$ is a set of accepting states.

Definition 6. *A deterministic TM (DTM) is a TM, where the transition $\rightarrow_{TM}$ is a function from $Q \times \Sigma_\Delta$ to $Q \times \Sigma_\Delta \times \{L, R\}$.*

The movement of a DTM is entirely determined by the current control state and the symbol read. A configuration of $\mathcal{M}$ is a member of the language $\Sigma_\Delta^*(Q \times \Sigma_\Delta)\Sigma_\Delta^*$. The transition between configurations $\mathcal{M}$ can be defined as $((q, a), (q', b, d)) \in \rightarrow_{TM}$ where the machine is in state q and the tape cell currently pointed to by the machine holds the value a, then the machine rewrites the value a with b, switches to state q', and moves the pointer to the left if $d = L$ and to the right if $d = R$. A configuration is an accepting configuration if the state $q_F \in Q_F$ is an accepting state. The machine is said to halt at a word w if every run of $\mathcal{M}$ starting from the configuration $(q_0,\$)\,w$ eventually reaches an accepting state ($q_F \in Q_F$). The machine is said to be stopping if it stops on every input word. For the case where there exists a computation of $\mathcal{M}$ that just stops in an accepting state with an input word w, we say that $\mathcal{M}$ accepts the input word w. Otherwise, it rejects w. The language $\mathcal{L}(\mathcal{M})$ accepted by $\mathcal{M}$ consists of all words $w \in \Sigma^*$ accepted by $\mathcal{M}$.

3.6.2 Complexity Measures

To determine the amount of time and space needed to solve a computational problem, an abstract (mathematical) model like the TM is used. Given a halting TM $\mathcal{M}$ and an input word w, the time required by a TM is the total number of state transitions (or steps) the machine makes before it stops and outputs the answer (yes or no). A TM $\mathscr{M}$ is said to operate within the time (or space) $f(n)$ if the time $\mathscr{M}$ spends on each input word of length n is at most $f(n)$. The function f is defined as $f\colon \mathbb{N} \to \mathbb{N}$ such that for each $n \in \mathbb{N}$ the maximum time (or space) required by $\mathscr{M}$ before it terminates on an input word w of length n. A decision problem $\mathscr{A}$ can be solved in time $f(n)$ if there exists a TM operating in time $f(n)$ that solves the problem. Since complexity theory is interested in classifying problems according to their difficulty, sets of problems are defined according to some criteria. For example, the set of problems solvable in time $f(n)$ on a DTM (resp. NTM) running in time $O(f(n))$ is denoted by:

$$\mathsf{NTIME(f(n))} = \{\mathscr{L} \mid \exists \mathscr{M} \in \mathsf{NTM} : \mathscr{M}\ decides\ \mathscr{L}\ within\ time\ f(n)\}$$

$$\mathsf{TIME(f(n))} = \{\mathscr{L} \mid \exists \mathscr{M} \in \mathsf{DTM} : \mathscr{M}\ decides\ \mathscr{L}\ within\ time\ f(n)\}$$

Similarly, we define the class of problems that are solvable by a TM that uses $O(f(n))$ space.

$$\mathsf{NSPACE(f(n))} = \{\mathscr{L} \mid \exists \mathscr{M} \in \mathsf{NTM} : \mathscr{M}\ decides\ \mathscr{L}\ within\ space\ f(n)\}$$

$$\mathsf{SPACE(f(n))} = \{\mathscr{L} \mid \exists \mathscr{M} \in \mathsf{DTM} : \mathscr{M}\ decides\ \mathscr{L}\ within\ space\ f(n)\}$$

An important feature to note is that time and space complexities differing only by some linear constant do not really matter.

Definition 7. *Let c be a non-negative real number and let $t : \mathbb{N} \to \mathbb{N}$ be a function such that $lim_{n\to\infty} \frac{t(n)}{n} = 0$. Then $\mathsf{SPACE}(t(n)) = \mathsf{SPACE}(c \cdot t(n))$ and $\mathsf{TIME}(t(n)) = \mathsf{TIME}(c \cdot t(n))$.*

The standard complexity classes for deterministic and nondeterministic TM are defined as:

(i) $\mathsf{P} = \bigcup_{k\geq 0} \mathsf{TIM}E(n^k)$ is the class of problems solvable in polynomial-time,

(ii) $\mathsf{PSPACE} = \bigcup_{k\geq 0} \mathsf{SPACE}(n^k)$ is the class of problems solvable in polynomial space,

(iii) $\mathsf{NP} = \bigcup_{k\geq 0} \mathsf{NTIME}(n^k)$ is the class of problems solvable by non-deterministic polynomial-time TM (NPTTM),

(iv) $\mathsf{NPSPACE} = \bigcup_{k\geq 0} \mathsf{NSPACE}(n^k)$ is the class of problems solvable by non-deterministic polynomial-space TM (NPSTM),

(v) $\mathsf{EXPTIME} = \bigcup_{k\geq 0} \mathsf{TIME}(2^{n^k})$ is the class of problems that are solvable in exponential-time k.

(vi) $\mathsf{NEXPTIME} = \bigcup_{k\geq 0} \mathsf{NTIME}(2^{n^k})$ is the class of problems that are solvable by non-deterministic exponential-time k.

The following containments are standard:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} = \mathsf{NPSPACE} \subseteq \mathsf{EXPTIME}$$

A decidable problem is said to be elementary if it is in exponential-time k for some $k \in \mathbb{N}$. Otherwise, it is said to be non-elementary. For a complexity class $\mathscr{C}$, we write $co\mathscr{C}$ for the set of problems whose complements are solvable in $\mathscr{C}$. For example, coNP is the set of problems whose complements are in NP.

3.7 Wrap up

In this chapter, we introduced some basic mathematical concepts and notations that we will use in this thesis. LTS and automata theory are introduced, as well as TM and standard definitions and results from computability and complexity theory. Bisimulation and equivalence are also introduced.

CHAPTER 4

Modal and Temporal Formalisms

In the last thirty years, many formalisms have been introduced for modeling and specifying untimed and DS. To verify the properties of these systems, it is necessary to formally capture the desired or undesired behavior. A formalism for expressing the behavioral properties of a DS should not only be able to express requirements about such observable properties, but should also be able to relate them over time. Temporal logics are popular formalisms for expressing properties of DS. There are several alternatives of temporal logics for specifying DS, such as Linear Temporal Logic (LTL) [Pnu77], Computational Temporal Logic (CTL) [CE82], CTL* [EH86], and modal temporal logics, such as the μ calculus [Koz83] and Hennessy-Milner logic (HML) [HM85, Mil89].

In this chapter, we recall some well-known models and formalisms for modeling and defining properties of DS. The chapter is organized as follows. In section 4.1, we introduce the modal and temporal formalisms used to specify DS. We also present the main theorems used by these formalisms. In section 4.2, we present the formalisms used to model DS.

4.1 Modal and Temporal Models

Modal logics are important formalisms in mathematical logic because they provide interesting tradeoffs between expressiveness and theoretical complexity. Temporal logics are a well-established formalism for the specification and verification of DS. Temporal logics are a subset of modal logics. Temporal logics (TL) can be divided into two categories, linear temporal logics, expressing linear properties, and branching temporal logics, expressing branching temporal properties. Temporal logics are interpreted over infinite traces. Linear Temporal Logic (LTL) [MP92] is a standard representation of the first category. In LTL, a formula is interpreted over a sequence of traces. Computational Temporal Logic (CTL) [CE82] is a standard representation of the second category. In CTL, a formula is interpreted over a tree of observable events. CTL* [EH86] contains formulas of both LTL and CTL. The μ calculus [Koz83] is a formalism with branching time operators, more expressive than CTL*, and allows the specification of ω-regular languages and sets of trees. Hennessy-Milner Logic (HML) [HM85, Mil89] is a modal predecessor of the μ calculus, with greater interest and impact in computer science, and especially in specifying properties of DS.

4.1.1 Linear Temporal Logic

LTL is interpreted over (linear) sequences of observations called traces. A trace σ over a set $\mathbb{P}$ of atomic propositions is a ω word (see section 3.2 for notation) $\sigma = \sigma_0\sigma_1 \cdots \in (2^{\mathbb{P}})^{\leq\omega}$ over the alphabet $2^{\mathbb{P}}$ of states. We let p range over $\mathbb{P}$. A state is a single observation, and the interpretation of a state σ as a subset of $\mathbb{P}$ is that the proposition p holds (is true) in state σ if $p \in \sigma$.

Definition 8. *The syntax of LTL is defined by the grammar:*

$$\phi ::= \ true \mid p \mid \phi_1 \ \vee \ \phi_2 \mid \neg\phi \mid \bigcirc\phi \mid \phi_1 \ \mathscr{U} \ \phi_2 \mid \phi_1 \ \mathscr{S} \ \phi_2$$

where $p \in \mathbb{P}$ is an atomic proposition, and ϕ_1, ϕ_2 are well-formed LTL formulas.

The semantics of LTL formulas is formally defined for a model (traces) $\sigma = \sigma_0\sigma_1\ldots$ and a formula ϕ by means of the satisfaction relation $\models$, as follows.

Definition 9. *The semantics of LTL formula ϕ is evaluated in a position $i \geq 0$ of a trace σ over $2^{\mathbb{P}}$ according to the following rules:*

$$\begin{array}{lcl}
(\sigma, i) \models & true & \textit{for every state in } \sigma \\
(\sigma, i) \models & p & \textit{iff } p \in \sigma_i, \ \textit{with } p \in \mathbb{P} \\
(\sigma, i) \models & \neg\phi & \textit{iff } (\sigma, i) \not\models \phi \\
(\sigma, i) \models & \phi_1 \vee \phi_2 & \textit{iff } (\sigma, i) \models \phi_1 \ \textit{or} \ (\sigma, i) \models \phi_2 \\
(\sigma, i) \models & \bigcirc\phi & \textit{iff } (\sigma, i+1) \models \phi \\
(\sigma, i) \models & \phi_1 \mathscr{U} \phi_2 & \textit{iff } \exists j \geq i, \textit{such that} (\sigma, j) \models \phi_2 \ \textit{and} \ \forall k, i \leq k < j, \\
 & & (\sigma, k) \models \phi_1 \\
(\sigma, i) \models & \phi_1 \mathscr{S} \phi_2 & \textit{iff } \exists j, 0 \leq j \leq i, \textit{such that} (\sigma, j) \models \phi_2 \ \textit{and} \ \forall k, j \leq k < i, \\
 & & (\sigma, k) \models \phi_1
\end{array}$$

$(\sigma, i) \models \phi$ denotes that the formula ϕ holds for the trace σ. The proposition p evaluates to the value of the atomic proposition p in the first state. The operators $true$, $\neg$, $\vee$ are propositional connectives. The last three operators relate states over time ($\bigcirc$, $\mathscr{U}$ $\mathscr{S}$). The formula $\bigcirc\phi$ holds for the trace obtained by removing the first state from σ. The operator is pronounced “next”. The formula $\phi_1\ \mathscr{U}\ \phi_2$ holds if there is some moment j (now or in the future) at which ϕ_2 is true (holds for (σ, j)) and for all moments before that ϕ_1 holds, i.e. for all k, $i \leq k < j$, $(\sigma, k) \models \phi_1$. The operator is pronounced “until”. The formula $\phi_1\ \mathscr{S}\ \phi_2$ holds if there is some moment j (now or before) at which ϕ_2 is true (holds for (σ, j)) and for all moments after that ϕ_1 holds, i.e. for all k, $j \leq k < i$, $(\sigma, k) \models \phi_1$. The operator is pronounced “since”.

Other operators can be defined in terms of these.

- false: false $\equiv \neg true$,
- disjunction: $\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$,
- the “eventually ϕ in the future” operator: $\Diamond\phi \equiv true\,\mathscr{U}\ \phi$, stating that ϕ will be true eventually (sometimes written as $F\phi$),
- the “always ϕ in the future” operator: $\Box\phi \equiv \neg\Diamond\neg\phi$, stating that ϕ is true at every moment (sometimes written as $G\phi$),
- the operator Release R is defined as the dual of the Until operator $\phi_1\mathscr{R}\phi_2 \equiv \neg(\neg\phi_1\mathscr{U}\neg\phi_2)$.
- the “eventually ϕ in the past” operator: $\blacklozenge\phi \equiv true\mathscr{S}\phi$, stating that ϕ was at some point true,
- the “always ϕ in the past” operator: $\blacksquare\phi \equiv \neg\blacklozenge\neg\phi$, stating that ϕ was true at every moment.

The language $\mathscr{L}(\phi)$ associated with the LTL formula ϕ (w.r.t. a set $\mathbb{P}$ of propositions) is the set of all traces over $\mathbb{P}$ that satisfy ϕ,

$$\mathscr{L}(\phi) = \left\{\phi \in \left(2^{\mathbb{P}}\right)^{\omega} \mid \sigma \models \phi\right\}$$

Example 1. *A trace over the set $\{p, q\}$ of propositions is for instance*

$$\{p\}\{p\}\{p, q\}\emptyset\{q\}\ldots$$

It satisfies the formula $p\ \mathscr{U}\ q$ (q holds at the third instant, and p for the first and second), but not the formula $\mathscr{G}p$ (p fails to hold at the fourth and fifth instant).

Example 2. *The formula $\Box(p \rightarrow \Diamond q)$, expresses that p propositions are followed by q propositions.*

Example 3. *The formula $\Box(p \rightarrow (p\ \mathscr{U}\ q))$, expresses that when p is true then it remains true until a q is reached.*

The LTL model checking problem is to establish whether TS $\models \phi$ holds for a given finite transition system TS (without terminal states) and the LTL state formula ϕ.

Theorem 1. *[SC85] The LTL model-checking problem is PSPACE-complete.*

The LTL satisfiability problem is to determine whether a given LTL formula ϕ is satisfiable. The LTL validity problem is to determine whether a given LTL formula ϕ is valid.

Theorem 2. *[SC85] The satisfiability and validity problems for LTL are PSPACE-complete.*

4.1.2 Hennessy-Milner Logic

Hennessy-Milner Logic (HML) is a modal logic that describes properties of states in a LTS over the set of actions or symbols Σ [HM85]. HML was created as an alternative approach to formalizing the notion of observational equivalence for concurrent and communication systems [Mil89].

Definition 10. *Let Σ be a finite alphabet, the set $\mathscr{H}$ of HML formulas over Σ is defined by the following grammar:*

$$\varphi ::= \ true \mid false \mid \varphi_1 \ \wedge \ \varphi_2 \mid \varphi_1 \ \vee \ \varphi_2 \mid [a]\varphi \mid \langle a \rangle \varphi$$

where $a \in \Sigma$, $\wedge$ and $\vee$ are boolean operators, and $[a]\varphi$, $\langle a \rangle\varphi$ are two modalities of the logic.

The semantics of the diamond modality $\langle a \rangle\varphi$ is, informally, that at a given state it is possible to execute an a symbol to a state where φ holds, and dually for the box modality $[a]\varphi$. The semantics of HML formulas is defined with respect to a given LTS. We can write formulas using abbreviations, i.e. if $\mathscr{F} = \{a_1, \ldots, a_n\} \subseteq \Sigma$, we can use the abbreviation $[\mathscr{F}]\varphi$ for the formulas $[a_1]\varphi \vee \ldots \vee [a_n]\varphi$ and $\langle \mathscr{F} \rangle \varphi$ for the formulas $\langle a_1 \rangle \varphi \wedge \ldots \wedge \langle a_n \rangle \varphi$. If $\mathscr{F} = \emptyset$ then $[\mathscr{F}]\varphi = false$ and $\langle \mathscr{F} \rangle \varphi = true$.

Definition 11. *Let Σ be a finite alphabet and let $\mathscr{S} = (S, s_0, \Sigma, \rightarrow_{lts})$ be a LTS over Σ. Let $\mathscr{H}$ be the set of HML formulas over Σ. The satisfiability relation $\models \subseteq (S \times \mathscr{H})$ relates states of the LTS $\mathscr{S}$ to the formulas they satisfy, and is defined as the smallest relation such that, for all states $s \in S$ and formulas φ, φ_1, $\varphi_2 \in \mathscr{H}$.*

$$\begin{array}{lcl}
s \models & true & \Rightarrow true \\
s \models & false & \Rightarrow false \\
s \models & \varphi_1 \vee \varphi_2 & \Rightarrow s \models \varphi_1 \ or\ s \models \varphi_2 \\
s \models & \varphi_1 \wedge \varphi_2 & \Rightarrow s \models \varphi_1 \ and\ s \models \varphi_2 \\
s \models & [a]\varphi & \Rightarrow \forall s' \in S \ such\ that\ s \xrightarrow{a}_{lts} s',\ then\ s' \models \varphi \\
s \models & \langle a \rangle \varphi & \Rightarrow \exists s' \in S \ such\ that\ s \xrightarrow{a}_{lts} s' \ and\ s' \models \varphi
\end{array}$$

One of the most interesting properties of HML is that it characterizes bisimilarity [HM85]. HML was defined as a formalism to understanding process equivalence in the context of Calculus of Communication Systems (CCS) [HM85] showed that if two CCS processes are bisimilar, then they satisfy the same set of HML formulas. A bisimulation equivalence between two LTS $\mathscr{H}_1$ and $\mathscr{H}_2$ with initial states $q0_{\mathscr{H}_1}$ and $q0_{\mathscr{H}_2}$, respectively, is an equivalence relation $\cong$ such that $q0_{\mathscr{H}_1} \cong q0_{\mathscr{H}_2}$ if, and only if, they satisfy the same set of HML formulas. Then, we say that the two states $q0_{\mathscr{H}_1}$ and $q0_{\mathscr{H}_2}$ (or equivalently the two initial states of the LTS $\mathscr{H}_1$ and $\mathscr{H}_2$) are bisimilar iff there is a bisimulation equivalence between them.

Due to the impossibility of expressing many temporal properties, more expressive modal logics have been studied. The μ-calculus logic [Koz83] is an extension of the HML with fixpoint operators.

4.1.3 Fixpoints

Fixpoints play an important role in computer science, especially in recursive algorithms. In denotational semantics, fixpoints are used to define the meaning of recursive definitions [Koz83]. Fixpoints are also used in set theory. Fixpoints can be computed by iterative recursion. Given a monotone function f whose domain is a complete lattice, we say that x is a fixpoint of f if $x = f(x)$, a pre-fixpoint of f if $f(x) \leq x$, and a post-fixpoint if $x \leq f(x)$.

Definition 12. *A partially ordered set (often abbreviated to poset) is a pair* $(\mathscr{A}, \leq)$*, where* $\mathscr{A}$ *is a set and* $\leq$ *is a reflexive, antisymmetric and transitive relation over* $\mathscr{A}$*.* $(\mathscr{A}, \leq)$ *is a totally ordered set if, for all* $a, e \in \mathscr{A}$*, either* $a \leq e$ *or* $e \leq a$ *holds.*

Definition 13. *Let* $(\mathscr{A}, \leq)$ *be a poset, and a set* $X \subseteq \mathscr{A}$*. Then,* $a \in \mathscr{A}$ *is an upper bound for* X *iff* $x \leq a$ *for all* $x \in X$*.* a *is the least upper bound (lub) of* X*, notation* $\triangle X$*, iff* a *is an upper bound for* X *and moreover,* $a \leq a'$ *for every* $a' \in \mathscr{A}$ *that is an upper bound for* X*. Also,* $a \in \mathscr{A}$ *is a lower bound for* X *iff* $a \leq x$ *for all* $x \in X$*.* a *is the greatest lower bound (glb) of* X*, notation* $\bigtriangledown X$*, iff* a *is a lower bound for* X *and moreover,* $a' \leq a$ *for every* $a' \in \mathscr{A}$ *that is a lower bound for* X*.*

Definition 14. *A poset* $(\mathscr{A}, \leq)$ *is a complete lattice iff* $\triangle X$ *and* $\bigtriangledown X$ *exist for every subset* X *of* $\mathscr{A}$*. A function* $f : \mathscr{A} \to \mathscr{A}$ *is a monotonic function iff* $a \leq a'$ *implies that* $f(a) \leq f(a')$ *for all* $a, a' \in \mathscr{A}$*.*

Theorem 3. *Let* $(\mathscr{A}, \leq)$ *be a complete lattice. Then* f *has a least fixpoint* x_μ *and a greatest fixpoint* x_ν *determined, respectively, by the pre-fixpoints and post-fixpoints of* f*:*

$$x_\mu = \triangle\{x \in \mathscr{A} \mid x \leq f(x)\},$$

$$x_\nu = \bigtriangledown\{x \in \mathscr{A} \mid f(x) \leq x\}$$

Definition 15. *Let* Σ *be a finite alphabet and let* $\mathscr{V}$ *be a set of variables. Then, the set* $\mathscr{M}$ *of* μ*-calculus formulas over* Σ *is defined by the following grammar:*

$$\varphi ::= true \mid false \mid Z \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a]\varphi \mid \langle a\rangle\varphi \mid \mu Z.\varphi \mid \nu Z.\varphi$$

where $a \in \Sigma$*,* $Z \in \mathscr{V}$*,* $\wedge$ *and* $\vee$ *are boolean operators,* $[a]\varphi$*,* $\langle a\rangle\varphi$ *are two modalities of the logic and* $\mu Z.\varphi$ *and* $\nu Z\varphi$ *are the minimal and maximal fixpoint operators of the logic.*

The semantics of μ-calculus formulas is defined with respect to a given LTS. We can write formulas using abbreviations, i.e. if $\mathscr{F} = \{a_1, \ldots, a_n\} \subseteq \Sigma$, we can use the abbreviation $[\mathscr{F}]\varphi$ for the formulas $[a_1]\varphi \vee \ldots \vee [a_n]\varphi$ and $\langle\mathscr{F}\rangle\varphi$ for the formulas $\langle a_1\rangle\varphi \wedge \ldots \wedge \langle a_n\rangle\varphi$. If $\mathscr{F} = \emptyset$ then $[\mathscr{F}]\varphi = false$ and $\langle\mathscr{F}\rangle\varphi = true$.

Definition 16. *Let* Σ *be a finite alphabet and let* $\mathscr{S} = (S, s_0, \Sigma, \rightarrow_{lts})$ *be a* LTS *over* Σ*. Let* $\mathscr{K} : \mathscr{V} \Rightarrow 2^S$ *be a valuation of variables. Let* $\mathscr{M}$ *be the set of* μ*-calculus formulas over* Σ*. The satisfiability relation* $\models \subseteq (S \times \mathscr{M})$ *relates states of the* LTS $\mathscr{S}$ *to the* *formulas they satisfy, and is defined as the smallest relation such that, for all states s ∈ S and formulas φ, φ_1, $\varphi_2 \in \mathscr{M}$.*

$$\begin{array}{lcl}
s \models & Z & \Rightarrow s \models \mathscr{K}(Z) \\
s \models & \varphi_1 \vee \varphi_2 & \Rightarrow s \models \varphi_1 \textit{ or } s \models \varphi_2 \\
s \models & \varphi_1 \wedge \varphi_2 & \Rightarrow s \models \varphi_1 \textit{ and } s \models \varphi_2 \\
s \models & [a]\varphi & \Rightarrow \forall s' \in S \textit{ such that } s \xrightarrow{a}_{lts} s',\ \textit{then } s' \models \varphi \\
s \models & \langle a \rangle \varphi & \Rightarrow \exists s' \in S \textit{ such that } s \xrightarrow{a}_{lts} s' \textit{ and } s' \models \varphi \\
s \models & \mu Z.\varphi & \Rightarrow s \models \varphi\{\mu Z.\varphi / Z\} \\
s \models & \nu Z.\varphi & \Rightarrow s \models \varphi\{\nu Z.\varphi / Z\}
\end{array}$$

A closed recursive formula of μ-calculus is a formula in which every formula variable Z is bound (i.e., every occurrence of Z appears within the scope of $\mu\ Z.\varphi$ or $\nu\ Z.\varphi$). A variable Z is free in the formula ϕ if some occurrence of it in ϕ is not bound. For example, the formula $\nu\ Z.Z$ is closed, but $\mu\ Z.[a]Y$ is not because Y is free in it. For formulas φ and ϕ, and a variable Z, $\varphi\{\phi/Z\}$ for the formula obtained by replacing every free occurrence of Z in φ with ϕ (e.g., $\varphi\{\phi/Z\}$).

4.2 ω-Automata

Finite Automata (FA) accept finite words (or finite traces) and can be adapted to accept infinite words (or infinite traces). FAs that accept infinite traces over finite alphabets are called ω automata [Büc62, Tho90]. Since the input trace never terminates (i.e., in ω automata), there are several notions of acceptance. The most common notion is Büchi acceptance (Büchi Automata (BA)). A run of a BA is accepting if it visits an infinite number of states marked as accepting. The Muller acceptance (Muller automata (MA)) [Tho90], replaces the set of accepting states by a set of sets of states.

4.2.1 Büchi Automata

In a BA the states are called locations. Formally, a BA is defined as follows.

Definition 17. *A BA is a tuple $\mathscr{A} = (Q, Q_0, \Sigma, \rightarrow_{BA}, Q_F)$ such that*

- *Q is the finite set of locations,*
- *$Q_0 \subseteq Q$ is the set of initial locations,*
- *Σ is the finite alphabet,*
- *$\rightarrow_{BA}$: $Q \times \Sigma \rightarrow 2^Q$ is the transition relation,*
- *$Q_F \subseteq Q$ is the set of acceptance location.*

Since a run of a BA is accepting if it visits locations in Q_F infinitely often (i.e., infinite trace (σ) over the alphabet Σ), it traverses locations of the BA while consuming symbols of σ. An accepting run of a BA on a trace (σ) is defined as:

Definition 18. *A run of a BA $\mathscr{A} = (Q, Q_0, \Sigma, \rightarrow_{BA}, Q_F)$ on a trace σ is an infinite sequence $\rho = q_0 q_1 \ldots q_n \ldots$ of locations such that:*

- *A run ρ is an initial location $q_0 \in Q_0$, and*

- *For all* $i \geq 0$, $q_{i+1} \in \rightarrow_{BA} (q_i, \sigma_i)$.

$\mathscr{A}$ accepts σ if $\mathscr{A}$ has an initial run on σ in which an accepting state occurs infinitely often (iff there exist infinitely many $i \geq 0$ such that $q_i \in Q_F$). The language recognized by $\mathscr{A}$ is the set of traces on which the automaton has an accepted run :

$$\mathscr{L}(\mathscr{A}) = \{\sigma \in \Sigma^\omega \mid there\ exists\ an\ accepting\ run\ for\ \sigma\ in\ \mathscr{A}\}.$$

We can say that $\mathscr{A}$ recognizes $\mathscr{L}(\mathscr{A})$. Normally, we can use a different notation for an accepting run for σ. We denote by $inf(\rho)$ the set of places that occur infinitely often in ρ. We say that $\mathscr{A}$ universally accepts σ if, for every initial run ρ of $\mathscr{A}$ on σ, $inf(\rho) \cap Q_F \neq \emptyset$. Languages recognized by non-deterministic BA (NBA) are called ω-regular languages. They are closed under all Boolean operations. The following ω-language of the NBA in Figure 4.1 is the set of traces ending with a^ω or ending with $(ab)^\omega$:

$$(\{a\} \cup \{b\})^* \{a\}^\omega \cup (\{a\} \cup \{b\})^* \{ab\}^\omega$$

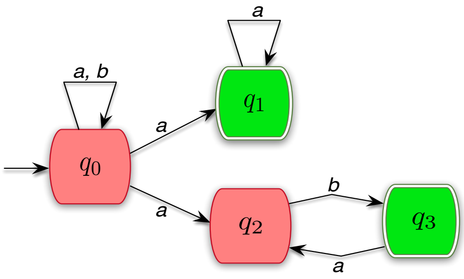


Figure 4.1: Nondeterministic Büchi Automaton

Nevertheless, the complement of this language (a^ω or $(ab)^\omega$) is recognized by the Deterministic BA (DBA) in Figure 4.2. The set of all languages recognized by DBA is therefore not closed under complementation. However, DBA is strictly less expressive than NBA. There are other conditions of acceptance for which DBA and NBA are equivalent.

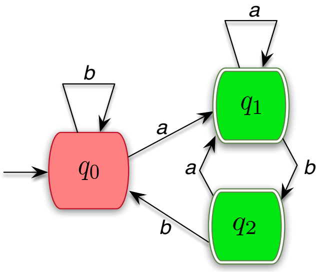


Figure 4.2: Deterministic Büchi Automaton

- Muller: The set of accepting locations Q_F is replaced by a set $\mathscr{Q}_F$ of sets of locations. A run ρ is then accepted if $inf(\rho) \in \mathscr{Q}_F$.
- Rabin: The set of accepting states Q_F is replaced by a set $\mathscr{F} = \{(E_1, Q_{F_1}), (E_2, Q_{F_2}), \ldots, (E_n, Q_{F_n})\}$ of pairs of sets of locations. A run ρ is then accepted if there is a pair (E_i, Q_{F_i}) in $\mathscr{F}$ such that $inf(\rho) \cap E_i = \emptyset$ *and* $inf(\rho) \cap Q_{F_i} \neq \emptyset$.
- Streett: This is the dual of the Rabin condition. A run ρ is accepted if, for every pair $(E_i, Q_{F_i}) \in \mathscr{F}$ $inf(\rho) \cap E_i = \emptyset \rightarrow inf(\rho) \cap Q_{F_i} \neq \emptyset$.
- Parity: The acceptance set Q_F is replaced by a colouring function, which assigns to every location some natural number $F\colon Q \times \mathbb{N}$. Then, ρ is accepted if $max\{F(q) \mid q \in inf(\rho)\}$ *is even*.

Definition 19. *The emptiness problem for a Büchi automaton $\mathscr{A}$ is to decide if the automaton $\mathscr{A}$ accepts at least one trace, i.e. to decide if $\mathscr{L}(\mathscr{A}) \neq \emptyset$*

Definition 20. *The universality problem for a Büchi automaton $\mathscr{A}$ is to decide if the automaton $\mathscr{A}$ accepts all possible traces on the set of Σ, i.e. to decide if $\mathscr{L}(\mathscr{A}) = \Sigma^{\leq\omega}$.*

The following theorems state that the emptiness and universality problems are decidable for BA and characterizes their complexity:

Theorem 4. *[SVW85] The problem of emptiness for BA is NLogSPACE-complete.*

Theorem 5. *[SVW85] The universality problem for BA is PSPACE-complete.*

Theorem 6. *[Büc62] Given two BA $\mathscr{A}$ and $\mathscr{B}$ defined on the same alphabet Σ, there exists a BA $\mathscr{C}$ defined on the alphabet Σ that accepts exactly the union of the language accepted by the two automaton $\mathscr{A}$ and $\mathscr{B}$, i.e. $\mathscr{L}(\mathscr{C}) = \mathscr{L}(\mathscr{A}) \cup \mathscr{L}(\mathscr{B})$. Further, the size of this automaton $\mathscr{C}$ is linear in the size of the automaton $\mathscr{A}$ and $\mathscr{B}$.*

Theorem 7. *[Büc62] Given two BA $\mathscr{A}$ and $\mathscr{B}$ defined on the same alphabet Σ, there exists a BA $\mathscr{C}$ defined on the alphabet Σ that accepts exactly the intersection of the language accepted by the two automaton $\mathscr{A}$ and $\mathscr{B}$, i.e. $\mathscr{L}(\mathscr{C}) = \mathscr{L}(\mathscr{A}) \cap \mathscr{L}(\mathscr{B})$. Further, the size of this automaton $\mathscr{C}$ is linear in the size of the automaton $\mathscr{A}$ and $\mathscr{B}$.*

Theorem 8. *[Büc62] Given a BA $\mathscr{A}$ over the alphabet Σ, there exists a BA $\mathscr{B}$ that accepts exactly the complement of language $\mathscr{A}$, i.e. $\mathscr{L}(\mathscr{B}) = \Sigma^{\leq\omega} \backslash \mathscr{L}(\mathscr{A})$. Further, the size of this automaton $\mathscr{B}$ is exponential in the size of the automaton $\mathscr{A}$.*

Definition 21. *A Labeled BA (LBA) is a tuple $\mathscr{A} = (Q, Q_0, \Sigma, \rightarrow_{BA}, \gamma, Q_F)$ such that*

- *Q is the finite set of locations.*
- *$Q_0 \subseteq Q$ is the set of initial locations,*
- *Σ is the finite alphabet ($\Sigma = 2^{\mathbb{P}}$),*
- *$\rightarrow_{BA} \subseteq Q \times Q$ is the transition relation.*
- *$\gamma : Q \rightarrow \Sigma$ is a function which labels each state $q \in Q$ with the set of atomic, propositions (Σ) are true in that state,*
- *$Q_F \subseteq Q$ is the subset of acceptance locations.*

Definition 22. *A run of a LBA $\mathscr{A} = (Q, Q_0, \Sigma, \rightarrow_{BA}, Q_F)$ on a trace σ is an infinite sequence $\rho = q_0 q_1 \ldots q_n \ldots$ of locations such that:*

- *The first location of the run ρ is a starting location $q_0 \in Q_0$, and*
- *for all $i \geq 0$, $(q_i, q_{i+1}) \in \rightarrow_{BA}$,*
- *The sequence σ respects the transition relation of $\mathscr{A}$, for all position $i \geq 0$ we have that $\sigma_i = \gamma(q_{(i)})$.*

Further, we say that the run ρ is accepting if it intersects the set of accepting places infinitely often, i.e. there are infinitely many places $i \geq 0$ such that $q_i \in Q_F$. We say that σ belongs to the language of $\mathscr{A}$, denoted $\mathscr{L}(\mathscr{A})$, if $\mathscr{A}$ has an accepted run on σ. The Figure 4.3 shows a LBA accepting an infinite number of a.

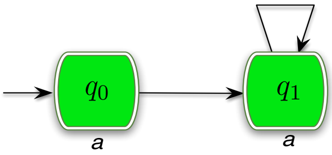


Figure 4.3: A LBA

4.3 Wrap up

In this chapter, we have introduced several modal and temporal formalisms that can be used to specify and reason about DS. We present the necessary background on modal logics, temporal logics, and BA formalisms.

Chapter 5

Distributed and Real-Time Temporal Formalisms

Temporal formalisms can be used in computer science to reason about properties of DS. Such properties are only qualitative, and it is impossible to express quantitative timing constraints on temporal events or actions. Thus, quantitative timing constraints are necessary to express properties of RTS and DRTS. Therefore, several modal and temporal formalisms have been extended to include quantitative timing constraints on temporal events (Timed Temporal Logics (TTL)). Some are extensions of LTL (Timed LTL (TLTL)), some adopt CTL (Timed CTL (TCTL)), and some are derived from modal and interval temporal logics. These logics can also be defined on discrete or dense temporal semantics. Some have been proposed as extensions of LTL, with quantitative time constraints and clocks in formulas (Metric Interval Temporal Logic (MITL) [AFH96], Timed Linear Temporal Logic (TPTL) [AFH96], L_v [LLW95], and Event Clock Temporal Logic EventClockTL [HRS98]). The complexity of TTL contrasts with the assumed quantitative timing constraints, and some logical extensions may be undecidable. Model checkers for RTS are Trex [ABS01], UPPAAL [UPP], Kronos [BDM+98] and Lash [LAS]. Model checkers with temporal formalisms that use dense temporal semantics can be more computationally expensive, and these can suffer more from the state space explosion problem. In this chapter, we recall some well-known models and formalisms for modeling and defining properties of RTS and DRTS. The chapter is organized as follows. Section 5.1, introduces the discrete and dense real-time semantics used to describe distributed and RTS. Section 5.2, presents the main real-time modal and temporal formalisms that have been proposed in the literature to specify distributed and real-time properties, and we recall their main features. Finally, section 5.3, presents the formalism of TA and some extensions that are used to model distributed and RTS.

5.1 Discrete and Dense Time Semantics

As mentioned in Section 2.6, there are two common ways to introduce quantitative time information into traces: **discrete time semantics** or **dense time semantics**. In the scientific literature, **discrete time semantics** extensively adopts natural and integer numbers ($\mathbb{N}$ and $\mathbb{Z}$). **Dense time semantics** use rational and real numbers ($\mathbb{Q}$ and $\mathbb{R}$).

Definition 23. *A time semantics is a structure* $(\mathbb{T}, \leq, 0, +)$ *satisfying the following properties:*

- $\mathbb{T}$ *is a finite or infinite set,*
- $\leq$ *is a total order relation on* $\mathbb{T}$*, defined as, for any* t_1*,* $t_2 \in \mathbb{T}$*,* $(t_1 \leq t_2) \Leftrightarrow$ *exists* $t_3 \in \mathbb{T}$*,* $t_3 \neq 0$ *and* $t_1 + t_3 = t_2$*,*
- *0 is the least element of* $\mathbb{T}$*,*
- $+$ *is an associative and commutative operator,*

In particular, the term time semantics refers to the set $\mathbb{T}$. Discrete and dense time semantics are mutually exclusive. An example for a discrete time semantics is the natural integers $(\mathbb{N}, \leq, 0, +)$. An example for a dense time semantics is the non-negative reals $(\mathbb{R}_{\geq 0}, \leq, 0, +)$.

- In a discrete time semantics, every point in time has a specific successor point: $\forall t_1 \in \mathbb{T}\ \exists t_2 \in \mathbb{T}, (t_1 < t_2 \text{ and } \forall t_3 \in \mathbb{T}\, t_1 < t_3 \Rightarrow t_1 \leq t_2)$,
- In a dense time semantics, every two points can find a point in time between these points: $\forall t_1, t_2 \in \mathbb{T},\ t_1 < t_2 \Rightarrow \exists t_3 \in \mathbb{T}, (t_1 < t_2 < t_3)$.

However, discrete and dense time semantics are commonly classified in terms of **pointwise** and **continuous** semantics. The **pointwise semantics** is evaluated along possibly infinite sequences of Timed Traces (TT):

Definition 24. *A Timed Trace (TT) over* Σ *is a sequence* $\rho = ((\sigma_0, t_0)(\sigma_1, t_1)\ldots(\sigma_n, t_n))$ *of actions or propositions paired with non-negative real numbers (i.e.,* $(\sigma_i, t_i) \in (\Sigma \times \mathbb{R}_{\geq 0})$*) such that the timestamp sequence* $t = t_0 \cdot t_1 \cdots t_n$ *is non-decreasing (i.e.,* $t_i \leq t_{i+1}$*). We sometimes define* ρ *as the pair* $\rho = (\sigma, t)$ *with* $\sigma \in \Sigma^*$ *and* t *a sequence of timestamps with the same length [AD94]. Furthermore, a timestamp sequence* $t = t_0 \cdot t_1 \cdots t_n$ *respects:*

(i) Initialization: $t_0 = 0$,
(ii) Monotonicity: for all $i \geq 0$, $t_{i+1} \geq t_i$,
(iii) Progress: for all $t \in \mathbb{R}_{\geq 0}$, *there exists* i *such that* $t_i > t$.

The **continuous semantics** is evaluated over possibly infinite signals: Given a set of propositions $\mathbb{P}$, a signal is a function $f : \mathbb{R}_{\geq 0} \rightarrow 2^{\mathbb{P}}$ mapping $t \in \mathbb{R}_{\geq 0}$ to the set $f(t)$ of propositions true at time t. A restriction of the continuous semantics for the evaluation of timed interval sequences is also called an interval-based semantics, or in other words, a continuous semantics with finite variability. A function f has the property of finite variability (also called non-Zenoness) if it can perform only finitely many actions or propositions in a finite time interval. The assumption is made to avoid the so-called Zeno's paradox, which means that infinitely many actions or propositions occur in a finite amount of time.

An interval I is a convex subset of the time semantics $\mathbb{T}$ (in this thesis, we will use $\mathbb{R}_{\geq 0}$). An interval is singular if it is a singleton. Two intervals I and I' are said to be adjacent when $I \cap I' = \emptyset$ and $I \cup I'$ is an interval. We denote by $\mathscr{I}_{\mathbb{R}_{\geq 0}}$ the set of intervals whose bounds are in $\mathbb{R}_{\geq 0}$. A finite interval I with lower bound l and upper bound r and l, $r \in \mathbb{R}_{\geq 0}$ is denoted as:

- $I = [l, r]$ $(\{t \in \mathbb{R}_{\geq 0} \mid l \leq t \leq r\})$ is both left and right closed interval and $l \leq r$,
- $I = [l, r)$ $(\{t \in \mathbb{R}_{\geq 0} \mid l \leq t < r\})$ is left closed and right open interval$l \leq r$,
- $I = (l, r]$ $(\{t \in \mathbb{R}_{\geq 0} \mid l < t \leq r\})$ is left open and right closed interval and $l < r$,
- $I = (l, r)$ $(\{t \in \mathbb{R}_{\geq 0} \mid l < t < r\})$ is both left and right open interval and $l < r$

The left bound of an interval I is denoted as $l(I)$ and the right bound of an interval I is denoted as $r(I)$. The length of interval I (i.e., $r - l$) is denoted as $|I|$. $I - t$ denotes the interval $I' = \{t' - t \mid t' \in I \; and \; t' \geq t\}$. An infinite interval I with lower bound a and no upper bound (i.e., ∞) is denoted as:

- $I = [l, \infty)$ $(\{t \in \mathbb{R}_{\geq 0} \mid l \leq t\})$ is left closed interval,
- $I = (a, \infty)$ $(\{t \in \mathbb{R}_{\geq 0} \mid l < t\})$ is left open interval.

An interval sequence over $\mathbb{R}_{\geq 0}$ is an infinite sequence $I = I_0 I_1 \cdots$ of non-empty intervals of $\mathscr{I}_{\mathbb{R}_{\geq 0}}$ where:

(i) successive intervals I_j and I_{j+1} are adjacent and $I_j < I_{j+1}$, for all $j \geq 0$
(ii) I is covering, i.e., for every $t \in \mathbb{R}^+$, there exists $j \in \mathbb{N}$ such that $t \in I_j$.

Definition 25. *A Timed Interval Sequence (TIS) is a pair* $\rho = (\sigma, I)$ *where* $\sigma = \sigma_0\sigma_1 \cdots$ *is an infinite sequence of states and* $I = I_0 I_1 \cdots$ *is an interval sequence. A TIS* ρ *can equivalently be seen as a sequence of elements in* $2^{\mathbb{P}} \times \mathscr{I}_{\mathbb{R}^+}$.

A graphical representation of a TIS $(\emptyset\{p\}\emptyset\ldots, [0,4)[4,6)[6\ldots)$ over $\mathbb{P} = \{p\}$ is shown in Figure 5.1.

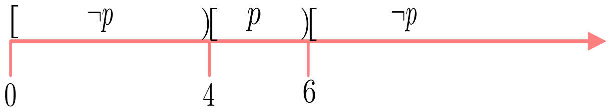


Figure 5.1: Example of a Timed Interval Sequence (TIS)

Now, our logics will admit infinite variability. Given two intervals I_1, I_2, we define the interval between I_1 and I_2 by $BetwI(I_1, I_2) = \{x \mid I_1 < x < I_2\}$. Given a set S and an interval I, we define *S Begins During I* by $\exists t \in (S \cap I)$, and $\nexists t' \in S$ such that $t' < I$. Symmetrically, we define *S Ends During I* iff $\exists\ t,\ t \in (S \cap I)$, and $\nexists\ t' \in S$ such that $t' > I$. In this thesis, we concentrate on the more interesting case when time is modeled by a dense time semantics. Here, we assume both **pointwise semantics** and **continuous semantics** with finite variability. We consider the real numbers here, but all the results presented in this thesis are also valid if the time semantics is the rational numbers.

5.2 Real-Time Modal and Temporal Logics

The following section deals with the extension of linear temporal logic with quantitative timing constraints.

5.2.1 Metric Temporal Logic

In this section, we define the syntax and semantics of Metric Temporal Logic (MTL) [AH93, Koy90]. MTL extends LTL by restricting the temporal operators to (bounded or unbounded) intervals of real numbers. For example, the formula $\Diamond_{[3,4]}\phi$ means that ϕ will be true in 3 to 4 time units from now. MTL extends LTL by adding time bounds to the Until and Since temporal operators.

Definition 26. *The syntax of MTL are defined by the grammar:*

$$\phi ::= \ true \mid p \mid \phi_1 \ \vee \ \phi_2 \mid \neg\phi \mid \phi_1 \ \mathcal{U}_I \ \phi_2 \mid \phi_1 \ \mathcal{S}_I \ \phi_2$$

where $p \in \mathbb{P}$ is an atomic proposition, I is an interval (that can be singular) and ϕ_1, ϕ_2 are well-formed MTL formulas.

Definition 27. *The MTL formula ϕ holds at the time $t \in \mathbb{R}_{\geq 0}$ of the TIS ρ, denoted $(\rho, t) \models \phi$, according to the following definition:*

$$\begin{array}{lcl}
(\rho,t) \models & p & \textit{iff } p \in \rho(t) \\
(\rho,t) \models & \phi_1 \vee \phi_2 & \textit{iff } (\rho,t) \models \phi_1 \textit{ or } (\rho,t) \models \phi_2 \\
(\rho,t) \models & \neg\phi & \textit{iff } (\rho,t) \not\models \phi \\
(\rho,t) \models & \phi_1 \mathcal{U}_I \phi_2 & \textit{iff} \exists t' \in (t+I) \textit{ with } (\rho,t') \models \phi_2, \textit{and} \,\forall\, t'' \in (t,t'), \\
 & & (\rho,t'') \models \phi_1 \\
(\rho,t) \models & \phi_1 \mathcal{S}_I \phi_2 & \textit{iff} \exists t' \in (t-I) \textit{ with} (\rho,t') \models \phi_2, \textit{and} \,\forall\, t'' \in (t',t), \\
 & & (\rho,t'') \models \phi_1
\end{array}$$

Example 4. *The following MTL formula $\Box(p \rightarrow \Diamond_{[5,5]}\, q)$ expresses that every event p is followed after exactly 5 unit of time by an event q.*

Unfortunately, over the dense time domain, the satisfiability and model checking problems for MTL are undecidable [Hen91]. This has led some researchers to consider various restrictions on MTL in order to recover decidability; see, e.g., [HMP92, Wil94, AD94]. Undecidability arises from the fact that MTL formulas can capture the computations of a Turing machine: configurations of the machine can be encoded within a single time interval of unit duration, since the density of time can accommodate arbitrarily large amounts of information. A MTL formula can then specify that the configurations be propagated exactly from one time interval to the next, so that the TIS satisfying the formula correspond exactly to the halting computations of the Turing machine.

Theorem 9. *[OW07] The logic MTL under the point-wise semantics is decidable, and non-primitive recursive complexity.*

Theorem 10. *[AFH96] The logic MTL under the continuous semantics is undecidable.*

5.2.2 Metric Interval Temporal Logic

Metric Interval Temporal Logic (MITL) [AFH96] is a restricted version of MTL in which the interval decorating the until and since modalities cannot be singular (i.e., reduced to a single point). MITL is a decidable variant that restricts temporal constraints on temporal operators to be in the form of an interval bounded by integers.

5.2.2.1 Pointwise Metric Interval Temporal Logic

The syntax and semantic of the pointwise MITL is based on timed traces (TT).

Definition 28. *The syntax of pointwise MITL is defined by the grammar:*

$$\phi ::= true \mid p \mid \phi_1 \vee \phi_2 \mid \neg\, \phi \mid \phi_1\, \mathcal{U}_I\, \phi_2 \mid \phi_1\, \mathcal{S}_I\, \phi_2$$

where $p \in \mathbb{P}$ is an atomic proposition, I is a nonsingular interval and ϕ_1, ϕ_2 are well-formed MITL formulas.

Definition 29. *The pointwise MITL formula ϕ holds in position $i \in \mathbb{N}$ of the TT $\rho = (\sigma, t)$, denoted $(\rho, i) \models \phi$, according to the following definition:*

$$\begin{array}{lcl}
(\rho,i) \models & p & \textit{iff } p \in \sigma_i \\
(\rho,i) \models & \phi_1 \vee \phi_2 & \textit{iff } (\rho,i) \models \phi_1 \textit{ or } (\rho,i) \models \phi_2 \\
(\rho,i) \models & \neg\phi & \textit{iff } (\rho,i) \not\models \phi \\
(\rho,i) \models & \phi_1 \mathcal{U}_I \phi_2 & \textit{iff } \exists j > i \textit{ such that } (\rho,i) \models \phi_2, \tau_j - \tau_i \in I \textit{ and } \forall k \textit{ with} \\
 & & i < k < j, \textit{we have}(\rho,k) \models \phi_1 \\
(\rho,i) \models & \phi_1 \mathcal{S}_I \phi_2 & \textit{iff } \exists j, 0 \le j < i \textit{ such that}(\rho,i) \models \phi_2, \tau_j - \tau_i \in I \textit{ and } \forall k \textit{ with} \\
 & & j < k < i, \textit{we have}(\rho,k) \models \phi_1
\end{array}$$

5.2.2.2 Continuous Metric Interval Temporal Logic

The formulas of MITL are built from propositional symbols, boolean connectives, and time-bound Until and Since operators.

Definition 30. *The syntax of* MITL *are defined by the grammar:*

$$\phi ::= true \mid p \mid \phi_1 \lor \phi_2 \mid \neg\, \phi \mid \phi_1 \,\mathcal{U}_I\, \phi_2 \mid \phi_1 \,\mathcal{S}_I\, \phi_2$$

where $p \in \mathbb{P}$ *is an atomic proposition,* I *is a nonsingular interval and* ϕ_1, ϕ_2 *are well-formed* MITL *formulas.*

Definition 31. *The* MITL *formula* ϕ *holds at time* $t \in \mathbb{R}_{\geq 0}$ *of the* TIS ρ*, denoted* $(\rho, t) \models \phi$*, according to the following definition:*

$$\begin{array}{lcl}
(\rho,t) \models & p & \textit{iff}\ p \in \rho(t) \\
(\rho,t) \models & \phi_1 \lor \phi_2 & \textit{iff}\ (\rho,t) \models \phi_1 \ \textit{or}\ (\rho,t) \models \phi_2 \\
(\rho,t) \models & \neg\phi & \textit{iff}\ (\rho,t) \not\models \phi \\
(\rho,t) \models & \phi_1 \mathcal{U}_I \phi_2 & \textit{iff}\ \exists t' \in (t+I)\ \textit{with}\ (\rho,t') \models \phi_2, \textit{and}\ \forall t'' \in (t,t'), \\
 & & \textit{we have}(\rho,t'') \models \phi_1 \\
(\rho,t) \models & \phi_1 \mathcal{S}_I \phi_2 & \textit{iff}\ \exists t' \in (t-I)\ \textit{with}(\rho,t') \models \phi_2, \textit{and}\ \forall t'' \in (t',t), \\
 & & \textit{we have}(\rho,t'') \models \phi_1
\end{array}$$

Below we also introduce a few derived temporal operators, more precisely, MITL derived operators if I is taken to be non-singular and ϕ is an MITL formula: $\Diamond_I \phi \equiv \top \mathcal{U}_I \phi$, $\Box_I \phi \equiv \neg \Diamond_I \neg \phi$. As well as their past counterparts: $\overleftarrow{\Diamond}_I \phi \equiv \bot \mathcal{S}_I \phi$, $\overleftarrow{\Box}_I \phi \equiv \neg \overleftarrow{\Diamond}_I \neg \phi$.

Example 5. *The following* MITL *formula* $\Diamond_{(0,5)} p$ *expresses that when evaluated in time* t*, eventually* p *holds in the interval* $t + (0,5)$.

Theorem 11. *[AFH96] The satisfiability and variability problems for* MITL *are decidable.*

Theorem 12. *[AFH96] The complexity of the satisfiability and validity problems for* MITL *are* $\mathsf{EXPSPACE}$*-*$\mathsf{Complete}$*.*

Theorem 13. *[AFH96] The satisfiability and variability problems for* $\mathsf{MITL}_{0,\infty}$ *are* PSPACE*-*$\mathsf{Complete}$*.*

5.2.3 Event Clock Temporal Logic

Here, we present Event Clock Temporal logic (EventClockTL), a logic introduced in [RS97]. The language expressible by EventClockTL formula can be defined by Event Clock Automata ECA [AFH94], a subclass of TA.

5.2.3.1 Point-wise Event Clock Temporal Logic

The syntax and semantic of the point-wise EventClockTL is based on timed traces (TT).

Definition 32. *The syntax of* $\mathsf{EventClockTL}$ *is defined by the grammar:*

$$\phi ::= true \mid p \mid \rhd_{\sim c}\, \phi \mid \lhd_{\sim c}\, \phi \mid \phi_1 \land \phi_2 \mid \phi_1 \lor \phi_2 \mid \neg\, \phi \mid \phi_1 \,\mathcal{U}\, \phi_2 \mid \phi_1 \,\mathcal{S}\, \phi_2$$

where $p \in \mathbb{P}$ is an atomic proposition, I is an interval (that can be singular), c is an integer constant, $\sim\, \in \{<, \leq, =, \geq, >\}$ and ϕ_1, ϕ_2 are well-formed EventClockTL formulas.

Definition 33. *The point-wise EventClockTL formula ϕ holds in position $i \in \mathbb{N}$ of the TT $\varrho = (\sigma, \tau)$, denoted $(\varrho, i) \models \phi$, according to the following definition:*

$$\begin{array}{lcl}
(\varrho,i)\models & p & \textit{iff}\ p \in \sigma_i \\
(\varrho,i)\models & \phi_1 \vee \phi_2 & \textit{iff}\ (\varrho,i)\models \phi_1\ \textit{or}\ (\rho,t)\models\phi_2 \\
(\varrho,i)\models & \phi_1 \wedge \phi_2 & \textit{iff}\ (\varrho,i)\models \phi_1\ \textit{and}\ (\rho,t)\models\phi_2 \\
(\varrho,i)\models & \neg\phi & \textit{iff}\ (\varrho,i)\not\models \phi \\
(\varrho,i)\models & \phi_1\mathcal{U}\phi_2 & \textit{iff}\ \exists j \in i\ \textit{such that}\ (\varrho,j)\models\phi_2, \textit{and}\ \forall k, j<k\leq i, \\
 & & \textit{we have}(\varrho,k)\models\phi_1 \\
(\varrho,i)\models & \phi_1\mathcal{S}\phi_2 & \textit{iff}\ \exists j, 0\leq j\leq i\ \textit{such that}(\varrho,j)\models\phi_2, \textit{and}\ \forall k, j\leq k\leq i, \\
 & & \textit{we have}(\varrho,k)\models\phi_1 \\
(\varrho,i)\models & \triangleright_{\sim c}\phi & \textit{iff}\ \exists j>i\ \textit{such that}\ (\varrho,j)\models\phi, \forall k, i<k<j, (\varrho,k)\not\models\phi \\
 & & \textit{and}\ \tau_j-\tau_i\sim c
\end{array}$$

$$\begin{array}{lcl}
(\varrho,i)\models & \triangleleft_{\sim c}\phi & \textit{iff}\ \exists j, 0\leq j<i\ \textit{such that}\ (\varrho,j)\models\phi,\ \forall k, j<k<i, (\varrho,k)\not\models\phi \\
 & & \textit{and}\ \tau_i-\tau_j\sim c
\end{array}$$

Example 6. *The following EventClockTL formula $\Diamond(\triangleright_{\leq 2} p)$ expresses that eventually p will be true within 2 time units.*

Example 7. *The following EventClockTL formula $\Box(\triangleleft_{=1} q \rightarrow \Diamond(\triangleright_{\leq 3} p))$ expresses that if q is exactly distant 1 time units of p, then eventually p will be true within 3 time units.*

Example 8. *The following EventClockTL formula $(p \wedge \triangleleft_{<2} q)$ expresses that if p occurs preceded, 2 time units before by a q.*

Theorem 14. *[Ras99] The satisfiability and variability problems for EventClockTL are decidable.*

Theorem 15. *[Ras99] The complexity of the satisfiability and validity problems for EventClockTL are PSPACE-Complete.*

5.2.3.2 Continuous Event Clock Temporal Logic

The formulas of EventClockTL are built from propositional symbols, boolean connectives, and time bounded until and since operators and two real-time operators at time t: the recording operator $\triangleleft_I\, \phi$ asserts that ϕ was true last time in the interval $t - I$, and the predicting operator $\triangleright_I\, \phi$ asserts that ϕ will be true next time in the interval $t + I$.

Definition 34. *The syntax of EventClockTL are defined by the grammar:*

$$\phi ::= true \mid p \mid \triangleright_I\ \phi \mid \triangleleft_I\ \phi \mid \phi_1\ \wedge\ \phi_2 \mid \phi_1\ \vee\ \phi_2 \mid \neg\ \phi \mid \phi_1\ \mathcal{U}_I\ \phi_2 \mid \phi_1\ \mathcal{S}_I\ \phi_2$$

where $p \in \mathbb{P}$ is an atomic proposition, I is an interval (that can be singular) and ϕ_1, ϕ_2 are well-formed formulas.

We can now define how to evaluate the truth value of an EventClockTL formula along TIS. Let ϕ be a continuous EventClockTL formula and let ρ be a TIS contain all propositions that occur in ϕ.

Definition 35. *The EventClockTL formula ϕ holds at time $t \in \mathbb{R}_{\geq 0}$ of the TIS ρ, denoted $(\rho, t) \models \phi$, according to the following definition:*

$$
\begin{array}{lcl}
(\rho,t) \models & p & \textit{iff}\, p \in \rho(t) \\
(\rho,t) \models & \phi_1 \vee \phi_2 & \textit{iff}\, (\rho,t) \models \phi_1 \textit{ or } (\rho,t) \models \phi_2 \\
(\rho,t) \models & \phi_1 \wedge \phi_2 & \textit{iff}\, (\rho,t) \models \phi_1 \textit{ and } (\rho,t) \models \phi_2 \\
(\rho,t) \models & \neg\phi & \textit{iff}\, (\rho,t) \not\models \phi \\
(\rho,t) \models & \phi_1 \mathscr{U}_I \phi_2 & \textit{iff}\, \exists t' \in (t+I) \textit{ with } (\rho,t') \models \phi_2, \textit{and}\, \forall t'' \in (t,t'), \\
 & & \textit{we have}(\rho,t'') \models \phi_1 \\
(\rho,t) \models & \phi_1 \mathscr{S}_I \phi_2 & \textit{iff}\, \exists t' \in (t-I) \textit{ with}(\rho,t') \models \phi_2, \textit{and}\, \forall t'' \in (t',t), \\
 & & \textit{we have}(\rho,t'') \models \phi_1 \\
(\rho,t) \models & \lhd_I \phi & \textit{iff}\, \exists t' < t \textit{ with } t' \in (t-I) \textit{ and } (\rho,t') \models \phi, \textit{and}\, \forall t'' < t, \\
 & & \textit{with } t'' > (t-I), (\rho,t'') \not\models \phi \\
(\rho,t) \models & \rhd_I \phi & \textit{iff }\, \exists t' > t \textit{ with } t' \in (t+I) \textit{ and } (\rho,t') \models \phi, \textit{and}\, \forall t'' > t \\
 & & \textit{with } t'' < (t+I), (\rho,t'') \not\models \phi
\end{array}
$$

As usual, we can define other temporal operators (LTL) and real-time operators. Note that they are defined in a strict way, i.e. they do not constrain the current state. Non-strict operators are easily defined from their strict counterparts. We use the following syntactic shortcuts for derived operators: $\top \equiv \neg\phi_1 \vee \phi_1$, and $\bot \equiv \neg\top$, $\Diamond\phi \equiv \top\mathscr{U}\phi$, $\Box\phi \equiv \neg\Diamond\neg\phi$. As well as their past counterparts: $\overleftarrow{\Diamond}\phi \equiv \bot\mathscr{S}\phi$, $\overleftarrow{\Box}\phi \equiv \neg\overleftarrow{\Diamond}\neg\phi$. Just for the behavior semantics, there are: $\mathscr{B}\phi \equiv \phi\mathscr{S}\phi$, $\mathscr{J}\phi \equiv \bot\mathscr{U}\phi$, $\overleftarrow{\mathscr{J}}\phi \equiv \bot\mathscr{S}\phi$.

Theorem 16. *[Ras99] The satisfiability and variability problems for EventClockTL are decidable.*

Theorem 17. *[Ras99] The complexity of the satisfiability and validity problem for EventClockTL are PSPACE-Complete.*

5.2.4 Timed Modal Logics

The Timed Modal Logic L_ν is a real-time extension of the Hennessy-Milner Logic (HML) with greatest fixed-points [LLW95]. L_ν is a modal logic that describe properties of states in a TLTS over the set of actions or symbols Σ. We now present the syntax of the logic.

Definition 36. *Let Σ be a finite alphabet of actions and X be a finite set of clocks, the formulae of L_ν over Σ, X and Id are defined by the grammar:*

$$\varphi ::= \ true \mid false \mid \varphi_1 \ \wedge \ \varphi_2 \mid \varphi_1 \ \vee \ \varphi_2 \mid [a]\varphi \mid \langle a \rangle \varphi \mid \exists\varphi \mid \forall\varphi \mid x \ \underline{in} \ \varphi \mid \phi \mid Z$$

where $a \in \Sigma$, $x \in X$, $\phi \in \Phi(X)$, $Z \in$ Id, $[a]\varphi$, $\langle a \rangle\varphi$ are two modalities of the logic, and $\exists$ φ and $\forall$ φ are the two time modalities.

The meaning of the identifiers in Id is specified by a declaration $\mathscr{D}$ assigning a L_v formula to every identifier in order to define properties with maximal fixpoints. A declaration is of the form $Z_i = \mathscr{D}(\overline{Z})$ where $\overline{Z} = (Z_1, \ldots, Z_n)$ and $\mathscr{D}(\overline{Z})$ is a formula over identifiers in $\overline{Z}$.

Definition 37. *Let Σ be a finite alphabet, X be a finite set of clocks. The semantics of formulas in L_v is implicitly given with respect to a given LTS inductively as follows:*

$$\begin{array}{lcl}
(q,\mu) \models & true & \Leftrightarrow true \\
(q,\mu) \models & false & \Leftrightarrow false \\
(q,\mu) \models & \varphi_1 \wedge \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \textit{ and } (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \varphi_1 \vee \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \textit{ or } (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \phi & \Leftrightarrow \mu \models \phi \textit{ for } \phi \in \Phi(X) \\
(q,\mu) \models & [a]\varphi & \Leftrightarrow \forall q \xrightarrow{a}_{lts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & \langle a \rangle \varphi & \Leftrightarrow \exists q \xrightarrow{a}_{lts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & x\underline{in}\varphi & \Leftrightarrow (q, \mu[x^p \to 0]) \models \varphi \\
(q,\mu) \models & \exists\varphi & \Leftrightarrow \exists\ t \in \mathbb{R}_{\geq 0}, \exists q' \in Q, \textit{ such that } q \xrightarrow{t}_{lts} q', (q,\mu+t) \models \varphi \\
(q,\mu) \models & \forall\varphi & \Leftrightarrow \forall\ t \in \mathbb{R}_{\geq 0}, \forall q' \in Q, \textit{such that } q \xrightarrow{t}_{lts} q', (q,\mu+t) \models \varphi \\
(q,\mu) \models & x + c \sim y + t & \Leftrightarrow \mu(x) + c \sim \mu(y) + t \\
(q,\mu) \models & Z & \textit{the greatest fixpoint in } \mathscr{D}(Z)
\end{array}$$

5.3 Formalisms of Distributed and Real-Time Automata

The following section deals with the extension of finite state automata with quantitative timing constraints.

5.3.1 Timed Labelled Transition Systems (TLTS)

Timed Labelled Transition Systems (TLTS) are used to describe the behavior (i.e., all possible executions) of a model of a RTS. TLTS can be used to describe the operational semantics of TA. Intuitively, a TLTS consists of a set of states and a set of transitions between states. Each transition denotes either a time delay or an action. Thus, a TLTS prescribes a set of allowed timed traces (TT).

Definition 38 (**Timed Labelled Transition Systems**)**.** *A TLTS [HMP91] is a tuple $\mathscr{D} = (Q, q_0, \Sigma, \rightarrow_{tlts}, Q_F)$ where:*

(i) Q is a finite set of states,

(ii) $q_0 \in Q$ is the initial state,

(iii) Σ is a finite alphabet of actions,

(iv) $\rightarrow_{tlts} \subseteq Q \times (\Sigma \uplus \mathbb{R}_{\geq 0}) \times Q$ is a set of transitions

(v) Q_F is a finite set of final states.

The transitions from one state to another state of a TLTS are noted in the following way:

(i) Discrete transition: (q, a, q') is denoted $q \xrightarrow{a} q'$, if $a \in \Sigma$ and $(q, a, q') \in \rightarrow_{tlts}$ and,

(ii) Delay transition: (q,d,q') is denoted $q \xrightarrow{d} q'$, if $d \in \mathbb{R}_{\geq 0}$ and $(q,d,q') \in \rightarrow_{tlts}$.

A run of $\mathscr{D}$ can be defined as a finite sequence of moves, where discrete and continuous transitions alternate: $\theta = q_0 \xrightarrow{d_0} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{d_1} q'_1 \xrightarrow{a_2} q_2 \ldots \xrightarrow{d_{n-1}} q'_{n-1} \xrightarrow{a_n} q_n$, where $\forall 0 \leq i \leq n$, $q_i \in Q$, $\forall j \geq 0$, $d_j \in \mathbb{R}_{\geq 0}$, $q'_j \in Q$ and $\forall k \geq 1$, $a_k \in \Sigma$. A run is initial if it starts in q_0. Thus, an initial path describes one execution of the system. A run is accepting if it starts from the initial state and ends in a final state. The duration of a run is the sum of the delays that occur along the run. The timed trace corresponding to ρ is $\rho = ((a_0,t_0),(a_1,t_1)\ldots,(a_n,t_n))$, where $t_i = \sum_{j=0}^{i-1} d_j$. A finite timed trace ρ is accepted by $\mathscr{D}$, called *timed language*, if and only if there exists an accepting run over $\mathscr{D}$ whose timed trace is ρ. The timed language of $\mathscr{D}$, denoted $\mathscr{L}(\mathscr{D})$, is defined as the set of all finite timed traces accepted by $\mathscr{D}$. Two TLTS $\mathscr{D}_1$ and $\mathscr{D}_2$ are timed language equivalent if $\mathscr{L}(\mathscr{D}_1) = \mathscr{L}(\mathscr{D}_2)$, that is they accept the same timed traces.

Moreover, we require the following standard properties for TLTS:

(i) Time-determinism: for all q, q', $q'' \in Q$ and for all $d \in \mathbb{R}_{\geq 0}$, if $q \xrightarrow{d} q'$ and $q \xrightarrow{d} q''$, then $q' = q''$,

(ii) 0-Delay: for all q, $q' \in Q$, $q \xrightarrow{0} q' \equiv q = q'$,

(iii) Additivity: for all q, q', $q'' \in Q$ and for any d, $d' \in \mathbb{R}_{\geq 0}$, if $q \xrightarrow{d} q'$ and $q' \xrightarrow{d'} q''$, then $q \xrightarrow{d+d'} q''$,

(iv) Continuity: for all q, $q' \in Q$ and for some $d \in \mathbb{R}_{\geq 0}$, if $q \xrightarrow{d} q'$ then for any d', d'' $\in \mathbb{R}_{\geq 0}$ such that $d = d' + d''$, there exists q'' such that $q \xrightarrow{d'} q'' \xrightarrow{d''} q'$.

5.3.2 Timed Automata

Timed Automata (TA) are an extension of finite automata (FA) with a finite set of clocks that evolve synchronously with time and allow measurement of delays [AD94]. TA model RTS by accepting timed words that specify when a stimulus or response occurs. Time is tracked by clocks that evolve at the same rate. They can be reset along the transitions, so that at any time a clock value represents the time elapsed since the last reset. Each transition specifies clock constraints, making the transition fireable only if the current clock values satisfy them. A location can also specify clock invariants, forcing the automata to leave it if it becomes false.

5.3.2.1 Clock Constraints

Let X be a finite set of variables ranging over $\mathbb{R}_{\geq 0}$, called *clocks*. Let $\Phi^+(X)$ be a set of clock constraints over X. A clock constraint $\phi \in \Phi^+(X)$ can be defined by the following grammar:

$$\phi := true \mid x \sim c \mid x - y \sim c \mid \phi_1 \wedge \phi_2$$

where $x, y \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, >, \leq, \geq, =\}$. The clock constraints of the form $true$, $x \sim c$ are called *non-diagonal constraints* and those of the form $x - y \sim c$ are called *diagonal constraints*. The set of non-diagonal constraints over X is denoted by $\Phi(X)$. Here we use the non-diagonal constraints as in [AD94], where the comparison between two clocks is not allowed [BLR05]. It is known that diagonal constraints do not add expressiveness to TA. [BPDG98](Section 4.2): diagonal constraints can be removed from TA [BLR05], but this removal leads to an exponential increase in the number of states, which is generally unavoidable.

5.3.2.2 Clock Invariants

Let X be a finite set of variables ranging over $\mathbb{R}_{\geq 0}$, called clocks. Let $\Delta(X)$ be a set of clock invariants. A clock invariant $\varphi \in \Delta(X)$ is a clock constraint of the following form:

$$\varphi := true \mid x < c \mid x \leq c \mid \varphi_1 \wedge \varphi_2$$

where $x \in X$, $c \in \mathbb{N}$. Clock invariants restrict the amount of time that can be spent in a given state without changing to the next state.

5.3.2.3 Clock Valuations

Given a finite set of clocks X, a clock valuation function, $v : X \to \mathbb{R}_{\geq 0}$ assigning to each clock $x \in X$ a non-negative value $v(x)$. We note $\mathbb{R}^X_{\geq 0}$ the set of all clock valuations. We note v_0 the mapping that associates 0 to each clock. For a time value $t \in \mathbb{R}_{\geq 0}$, we note $v + t$ the valuation defined by $(v + t)(x) = v(x) + t$. Given a clock subset $Y \subseteq X$, we note $v[Y \leftarrow 0]$ the valuation defined as follows: $v[Y \leftarrow 0](x) = 0$ if $x \in Y$ and $v[Y \leftarrow 0](x) = v(x)$ otherwise. Given a clock constraint $\phi \in \Phi(X)$ and a clock valuation v, we say that v satisfies ϕ, denoted by $v \models \phi$ and defined formally as follows:

$$v \models x \sim c \Longleftrightarrow v(x) \sim c$$

$$v \models \phi_1 \wedge \phi_2 \Longleftrightarrow v(x) \models \phi_1 \wedge v(x) \models \phi_2$$

$$v \models true \Longleftrightarrow true$$

Given $Y \subseteq X$, the projection of v on Y, written $v\rfloor_Y$, is the valuation over Y only containing the values in v of clocks in Y. In the same way, it is also possible to say that for a clock invariant $\varphi \in \Delta(X)$ and a clock valuation v, v satisfies φ, denoted by $v \models \varphi$.

Definition 39 (Timed Automata). *A TA is a tuple $\mathscr{A} = (S, s_0, \Sigma, X, \to_{ta}, Inv, F)$ where:*

(i) S is a finite set of locations,
(ii) $s_0 \in S$ is the initial location,
(iii) Σ is a finite alphabet,
(iv) X is a finite set of clock names,
(v) $\to_{ta} \subseteq S \times \Sigma \times \Phi(X) \times 2^X \times S$ is the finite transition relation,
(vi) $Inv : S \to \Delta(X)$ associates to each location a clock invariant,
(vii) $F \subseteq S$ is a finite set of final locations.

For a transition $(s, a, \phi, Y, s') \in \rightarrow_{ta}$, we classically write $s \xrightarrow{a,\phi,Y} s'$ and call s and s' the source and target locations, ϕ the guard, a the action or label, Y the set of clocks to be reset. To execute a TA, we need to track clocks valuations. Furthermore, only legal locations are visitable, i.e., locations that satisfy $v \models Inv(s)$ (i.e. valuations that map clocks to values that satisfy the current state of the invariant). During the execution of a TA $\mathscr{A}$, a state is a pair $(s, v) \in S \times \mathbb{R}^X_{\geq 0}$, where s denotes the current location with its accompanying clock valuation v, starting at (s_0, v_0) where v_0 maps each clock to 0.

Definition 40 (**Semantics of TA**). *The semantics of a TA $\mathscr{A}$ is given by a TLTS($\mathscr{A}$) = $(Q, q_0, \Sigma, \rightarrow_{tlts}, Q_F)$ where $Q = \{(s, v) \in S \times \mathbb{R}^X_{\geq 0} \mid v \models Inv(s)\}$ is the set of states over $\mathscr{A}$ (with initial state $q_0 = (s_0, v_0)$), $Q_F = \{(s_f, v) \in F \times \mathbb{R}^X_{\geq 0} \mid v \models Inv(s_f)\}$ is the set of final states over $\mathscr{A}$ and $\rightarrow_{tlts} \subseteq Q \times (\Sigma \uplus \mathbb{R}_{\geq 0}) \times Q$ is the TLTS transition relation defined by:*

(i) Discrete transition: $(s, v) \xrightarrow{a} (s', v')$, such that $s \xrightarrow{a,\phi,Y} s'$, $v \models \phi$, $v' = v[Y \leftarrow 0]$ and $v' \models Inv(s')$,

(ii) Delay transition: $(s, v) \xrightarrow{t} (s, v + t)$ for any $t \in \mathbb{R}_{\geq 0}$, such that $v + t \models Inv(s)$.

A run θ of $\mathscr{A}$ is a finite sequence of consecutive delay and discrete transitions. The run is said to be accepting if it starts in s_0 with valuation v_0 and ends in a final location ($s_f \in F$). A run of $\mathscr{A}$ is a path ρ of TLTS($\mathscr{A}$) starting from the initial state $q_0 = (s_0, v_0)$, with delay and discrete transitions alternating along the run: $\theta = (s_0, v_0) \xrightarrow{t_0} (s_0, v'_0) \xrightarrow{a_1} (s_1, v_1) \xrightarrow{t_1} (s_1, v'_1) \xrightarrow{a_2} (s_2, v_2) \ldots \xrightarrow{t_{n-1}} (s_{n-1}, v'_{n-1}) \xrightarrow{a_n} (s_n, v_n)$ where $v_0(x) = 0$ for every $x \in X$. The timed trace associated with ρ (defined above) is $\rho = ((\sigma_0, t_0)(\sigma_1, t_1) \ldots (\sigma_n, t_n))$ where $(\sigma_i, t_i)_{0 \leq i \leq n} \in (\Sigma \times \mathbb{R}_{\geq 0})^{n+1}$. Conversely, we say that the run θ reads ρ in $\mathscr{A}$.

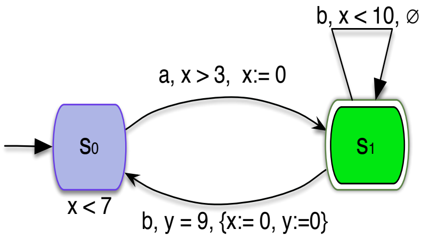


Figure 5.2: A TA

Example 9. *Let $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F)$ be the TA depicted in Figure 5.2. $\mathscr{A}$ contains two locations: l_0 (initial) and l_1. In particular, s_0 is the only location to define an invariant: $I(s_0) = (x < 7)$, forcing the TA to exit s_0 when x becomes smaller than 7. s_1 has the invariant true (by default, not drawn by I), allowing time to progress unboundedly while being in s_1. Suppose the current location is s_1. The transition $s_1 \xrightarrow{b,(y=9),\{x:=0;y:=0\}} s_0$ specifies that when the action b occurs and the guard $y = 9$ holds, this enables the transition, leading to a new current location s_0, while resetting* *clock variables x and y. Note that using a location invariant (which specifies when it may be possible to stay in a given location) is different from putting a constraint as a guard (which specifies that the transition is enabled only when the constraint holds).*

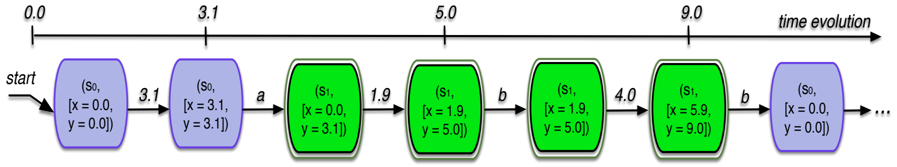


Figure 5.3: The run of the timed automaton $\mathscr{A}$ in Figure 5.2

Example 10. *Figure 5.3 presents a run of the timed automaton $\mathscr{A}$ presented in Figure 5.2. A run of $\mathscr{A}$ is a path in TLTS($\mathscr{A}$) = $(s_0,[x=0.0,y=0.0]) \xrightarrow{3.1} (s_0,[x=3.1,y=3.1]) \xrightarrow{a} (s_1,[x=0.0,y=3.1]) \xrightarrow{1.9} (s_1,[x=1.9,y=5.0]) \xrightarrow{b} (s_1,[x=1.9,y=5.0]) \xrightarrow{4.0} (s_1,[x=5.9,y=9.0]) \xrightarrow{b} (s_0,[x=0.0,y=0.0])$ starting from the initial state with each clock set to 0 and the timed trace θ associated with this non-accepting run is* $((a,3.1)(b,5.0)(b,9.0))$.

We write $\mathscr{L}(\mathscr{A})$ for the language of $\mathscr{A}$, that is the set of timed traces associated with an accepting run ($\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathsf{TLTS}(\mathscr{A}))$).

Theorem 18. *[AD94] TA are closed under union and intersection.*

But unfortunately TA are not closed under complement. Let us consider the TA $\mathscr{A}$ in Figure 5.4 to get some intuition about the reason why TA are not closed under complement. This TA accepts exactly the TT where two a are separated by a one-time unit. At location s_0, when reading a, the automaton can non-deterministically decide to stay at s_0 or to go to location s_1 and reset the clock x to 0.

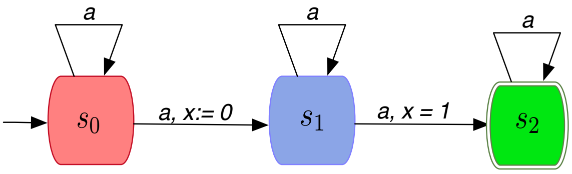


Figure 5.4: A Non complementable TA

Theorem 19. *[AD94] TA are not closed under complement.*

The emptiness problem for TA is to decide whether a given timed automaton $\mathscr{A}$ accepts at least one TT, that is, if $\mathscr{L}(\mathscr{A}) \neq \emptyset$.

Theorem 20. *[AD94] The emptiness problem for TA is decidable.*

Theorem 21. *[AD94] The emptiness problem for TA is PSPACE-Complete.*

Theorem 22. *[AD94] The problem of universality for TA is undecidable.*

The proof of the last theorem [AD94] shows that the problem of deciding whether a two-counter machine has a recurring computation can be reduced to the universality problem of TA. Since the recurring computation problem for two-counter machines is undecidable [FL79], it follows that the universality problem for TA is undecidable.

5.3.3 Region Equivalence and Region Automaton

A classical construction for the analysis of TA is the region automaton. Since the number of states in a TA is infinite, it is impossible to build a finite state automaton. Then, the region automaton abstracts the execution of a TA by merging states that are considered equivalent [AD94].

Given $t \in \mathbb{R}_{\geq 0}$, we note $\lfloor t \rfloor$ and $fract(t)$ the integral and fractional part of t, respectively. For a given TA $\mathscr{A}$, we note $c_x \in \mathbb{N}$ the maximal constant to which clock $x \in X$ is compared in the guards and invariants of $\mathscr{A}$.

Definition 41 (**Clock Equivalence**). *Two clock valuations v, $v' \in v \in \mathbb{R}^X_{\geq 0}$, are clock-equivalent (denoted by $v \equiv v'$) iff the following conditions are satisfied:*

(i) $\forall x \in X$, $v(x) > c_x$ if and only if $v'(x) > c_x$,

(ii) $\forall x \in X$, $v(x) \leq c_x$ implies both $\lfloor v(x) \rfloor = \lfloor v'(x) \rfloor$ and $fract(v(x)) = 0$ if and only if $fract(v'(x)) = 0$, and

(iii) $\forall x, y \in X$, such that $v(x) \leq c_x$ and $v(y) \leq c_y$, we have $fract(v(x)) \leq fract(v(y))$ if and only if $fract(v'(x)) \leq fract(v'(y))$.

The equivalence class generated by $\equiv$ is called a clock region, and we note $[v]$ the clock region that contains v, and $CRegions(\mathscr{A})$ the set of clock regions for $\mathscr{A}$.

Region equivalence extends to states by defining $q_{\mathscr{A}} \equiv q'_{\mathscr{A}}$, where $q_{\mathscr{A}}$, $q'_{\mathscr{A}}$ are states in TLTS($\mathscr{A}$) and $q_{\mathscr{A}} = (s_{\mathscr{A}}, v)$, $q'_{\mathscr{A}} = (s'_{\mathscr{A}}, v')$, iff (1) $s_{\mathscr{A}} = s'_{\mathscr{A}}$ (i.e., $s_{\mathscr{A}}$ and $s'_{\mathscr{A}}$ both are the same location), (2) $v \equiv v'$. Regions are the equivalence classes induced by $\equiv$ on the set of states.

Example 11. *For example, consider a TA with two clocks x and y with a maximal constant that is supposed to be 2 (i.e., $c_x = 2$ and $c_y = 2$). The partitions depicted in Figure 5.5(a) represent all constraints defined between constants, c_x smaller than or equal to 2 and c_y smaller than or equal to 2. Let us consider two valuations v_1 and v_2 which are not equivalent due to time elapsing (i.e., the possible behaviors from v_1 and v_2 are different). Condition 3 (Definition 41) refines the partition by adding diagonal lines which represent time elapsing (see Figure 5.5(b)).*

Definition 42 (**Region Automaton**). *Let $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F)$ be a TA. The region automaton $RG(\mathscr{A})$ of $\mathscr{A}$ is the transition system $(Q, q_0, \Sigma, \rightarrow_{lts}, Q_F)$ where states in Q are pairs (s, r) so-called symbolic states, where $s \in S$ and $r \in CRegions(\mathscr{A})$ (with initial state $q_0 = (s_0, [v_0])$ and final states $Q_F = \{(s, r) | \ s \in F\}$), and the transition relation $\rightarrow_{lts}$ is defined as follows:*

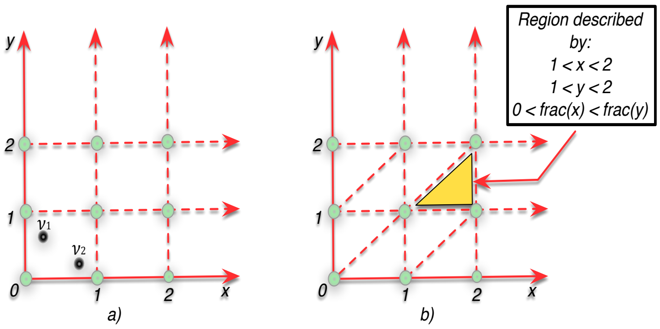


Figure 5.5: Two clock valuations v_1 and v_2 (a) and Region with clocks x, y and constant 2 (b)

(i) *Discrete transition: For all* $a \in \Sigma \cup \{\epsilon\}$, *there is a transition* $(s, r) \xrightarrow{a} (s', r')$, *if, and only if, there is a* $\mathscr{A}$*-transition* $s \xrightarrow{a,\phi,Y} s'$, *such that* $v \models \phi$ *and* $v' = v[Y \leftarrow 0]$ *with* $v \in r$ *and* $v' \in r'$.

(ii) *Delay transition: There is a transition* $(s, r) \xrightarrow{\epsilon} (s, r')$ *if, and only if, there is a* $t \in \mathbb{R}_{\geq 0}$ *and an* $\mathscr{A}$*-transition* $(s, v) \xrightarrow{t} (s, v + t)$, *if* $\forall\ t'\ 0 \leq t' \leq t$, $v + t' \models Inv(s)$, *with* $v \in r$ *and* $v + t \in r'$.

Hence, given a TA $\mathscr{A}$ and its region automaton RG($\mathscr{A}$), we can reduce the emptiness check for the timed language accepted by $\mathscr{A}$ ($\mathscr{L}(\mathscr{A})$) to a reachability problem in RG($\mathscr{A}$).

Theorem 23. *[AD94] Checking the reachability of a location in a timed automaton is a PSPACE-complete problem.*

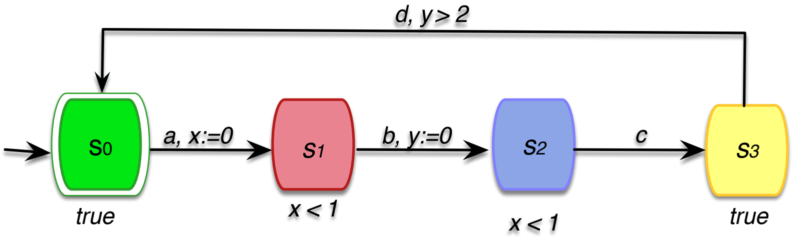


Figure 5.6: TA $\mathscr{A}$

Example 12. *Let us consider a TA* $\mathscr{A}$ *depicted in Figure 5.6 (taken from [AD94]) and region partition depicted in Figure 5.7. The corresponding region automaton RG(*$\mathscr{A}$*) is depicted in Figure 5.8. However, only the regions reachable from the initial region* $(s_0, x = y = 0)$ *are shown. In this example, the location* s_3 *of* $\mathscr{A}$ *is reachable if and only if one of the states* (s_3, r) *with a region* r *is reachable in the finite automaton given in Figure 5.6. In RG(*$\mathscr{A}$*), the path* $(s_0, x = y = 0) \xrightarrow{a} (s_1, 0 = x < y < 1) \xrightarrow{\epsilon} (s_1, 0 < x < y < 1)$

$\xrightarrow{b} (s_2, 0 = y < x < 1) \xrightarrow{\epsilon} (s_2, y < x < 1) \xrightarrow{c} (s_3, 0 < y < x < 1)$ *leads to location* s_3 *which implies that, in the TA* $\mathscr{A}$*, there is an execution* $(s_0, v_0) \xrightarrow{t_1} (s_0, v'_0) \xrightarrow{a} (s_1, v_1) \xrightarrow{t_2} (s_1, v'_1) \xrightarrow{b} (s_2, v_2) \xrightarrow{t_3} (s_2, v'_2) \xrightarrow{c} (s_3, v_3)$ *leading to* s_3 *(for some* t_1, t_2, $t_3 \in \mathbb{R}_{\geq 0}$*).*

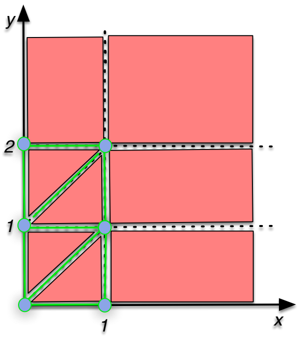


Figure 5.7: Clock Region

Contrary to the TLTS associated to a TA, which is potentially infinite, thus impossible to construct explicitly, the region automaton is finite: the number of clock regions is bounded by $|CRegions(\mathscr{A})| \leq \prod_{x \in X}((2c_x+2) \times |X|! \times 2^{|X|})$, leading to a number of states for the region automaton bounded by $((2c_x+2)^{|X|} \times |X|! \times 2^{|X|} \times |S|)$ [AD94].

5.3.4 Reachability Problem

The reachability problem is decidable for TA [AD94]. It asks if there exists a path from its initial state to a given target state. Two algorithms can be used to solve the reachability problem::

- Forward Analysis Algorithm: The goal of this algorithm is to iteratively compute the successors of the initial states and to check whether the state to be reached is finally computed or not.
- Backward Analysis Algorithm: This algorithm aims to iteratively compute the predecessors of the states to be reached and check whether the state to be reached is eventually computed or not.

The reachability problem for TA is decidable using the region automaton [AD94], but the number of regions is usually very large, making this solution impractical. To avoid using regions and get a more reasonable number of transitions, it is possible to use a convex union of regions called *clock zones* [BY04]. This approach is usually more efficient. It is implemented in classic tools like UPPAAL [UPP], KRONOS [BDM+98], TEMPO Toolset. [GMP13].

5.3.4.1 The Zone Approach

In a zone graph [BY04], clock zones are used to symbolically represent sets of clock valuations. To formally define the notion of a clock zone, over a set of clocks X, we

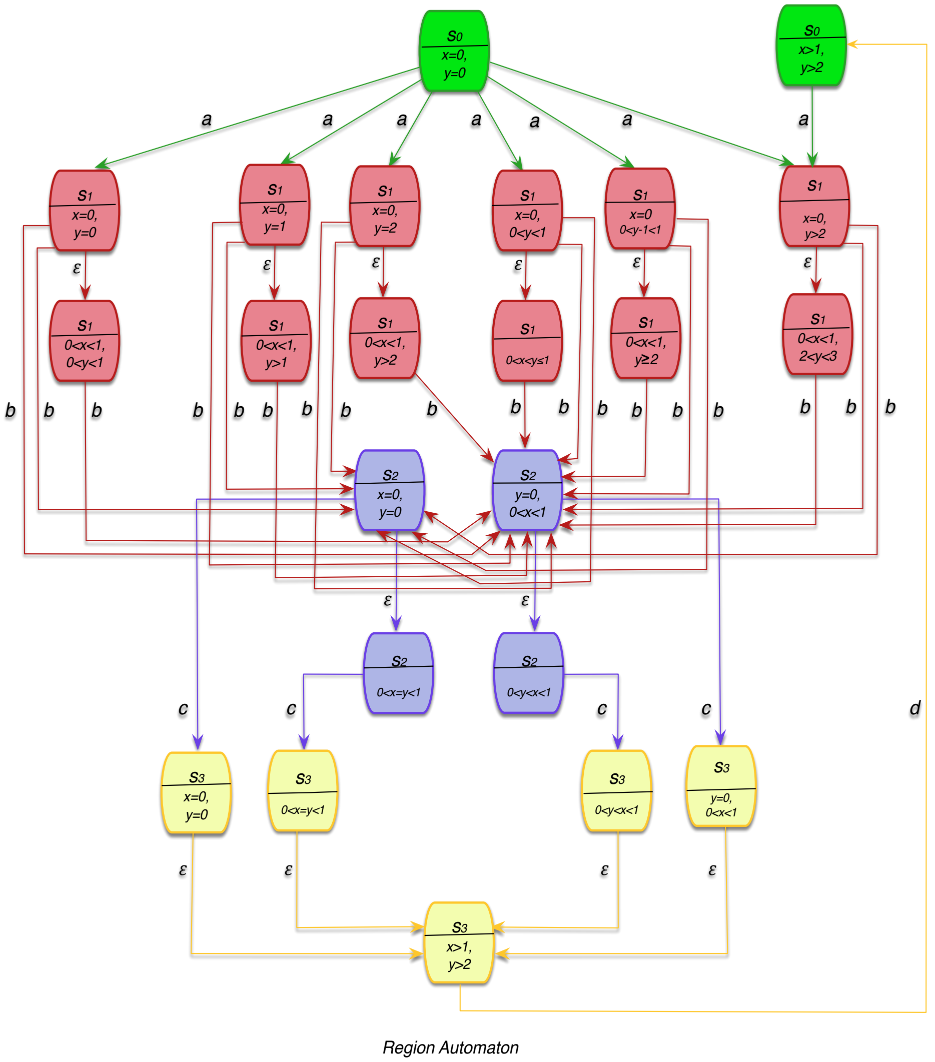


Figure 5.8: Region automaton associated with the TA $\mathscr{A}$ (Figure 5.6)

need to consider the set $\Phi^+(X)$ of diagonal clock constraints. Therefore, a clock zone $\mathcal{Z}$ is a convex set of clock valuations described by a diagonal constraint $\phi \in \Phi^+(X)$ such that $\mathcal{Z} = \{v \mid v \models \phi\}$. To have a unified representation for clock zones, a reference clock x_0 is introduced to the set of clocks X, which is always 0. The general form of a clock zone can be described by the following form:

$$\mathcal{Z} = (x_0 = 0) \wedge \bigwedge_{0 \le i \ne j \le n} x_i - x_j \preceq c_{i,j}$$

where x_i and x_j are clocks of X, $c_{i,j}$ is a non-negative integer and $\preceq \in \{\le, <\}$.

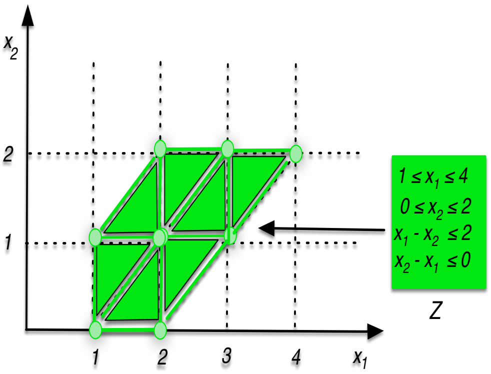


Figure 5.9: Clock zone $\mathcal{Z}$

Example 13. *Figure 5.9 presents the clock zone* $\mathcal{Z}$ = [[$x_1 \geq 1 \wedge x_1 \leq 4 \wedge x_2 \geq 0 \wedge x_2 \leq 2 \wedge x_1 - x_2 \leq 2 \wedge x_2 - x_1 \leq 0$]].

Therefore, the forward and backward analysis algorithms described above have been implemented using symbolic states [BBFL03]. A symbolic state is a pair $(s, \mathcal{Z})$, called *zones*, where s is a location and $\mathcal{Z}$ is a clock zone.

Definition 43. *Let* $\mathcal{Z}, \mathcal{Z}'$ *be two clock zones and* $Y \subseteq X$ *be a finite set of clocks. The semantics of the intersection, restricting projection, clock reset, inverse clock reset, time successor and time predecessor on a clock zone can be defined:*

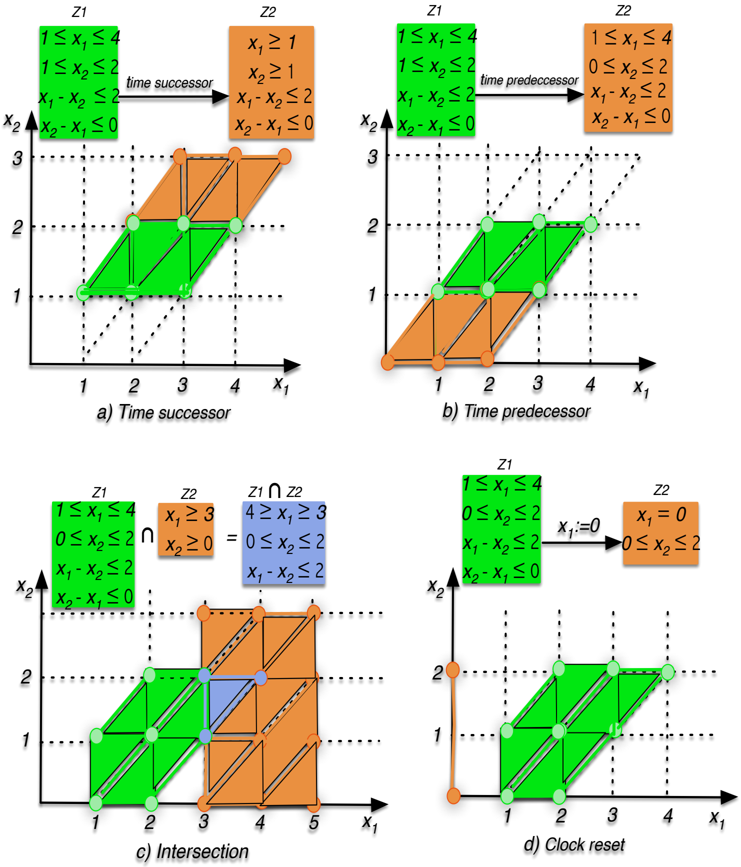
(i) Intersection of two clock zones, defined by:
- $\mathcal{Z} \cap \mathcal{Z}' = \{v \mid v \in \mathcal{Z} \wedge v \in \mathcal{Z}'\}$,

(ii) Restricting projection of a zone, defined by:
- $\mathcal{Z}\rfloor_Y = \{v \mid \exists v' \in \mathcal{Z} \wedge \forall x \in Y,\ v(x) = v'(x)\}$,

(iii) Clock reset and inverse clock reset of a zone defined by:
- $\mathcal{Z} \downarrow_Y = \{v[Y \leftarrow 0] \mid v \in \mathcal{Z}\}$,
- $\mathcal{Z} \uparrow_Y = \{v \mid v[Y \leftarrow 0] \in \mathcal{Z}\}$,

(iv) time successor and predecessor of a zone, defined by:
- $\mathcal{Z} \uparrow = \{v + t \mid v \in \mathcal{Z} \text{ and } t \in \mathbb{R}_{>0}\}$,
- $\mathcal{Z} \downarrow = \{v - t \mid v \in \mathcal{Z} \text{ and } t \in \mathbb{R}_{>0}\}$.

Notice that the operations time successor, time predecessor, clock reset and its inverse, and the standard intersection preserve clock zones [BY04] [AD94]. Some examples of the operations are presented in Figure 5.10. Given a TA $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F)$, it is possible to construct a zone graph ZG($\mathscr{A}$) such that states of ZG($\mathscr{A}$) are zones of $\mathscr{A}$. A zone graph ZG($\mathscr{A}$) [LLW95] is similar to the region graph [AD94] with the difference that each node consists of zones. Given a discrete transition e=$(s, a, \phi, Y, s') \in \rightarrow_{ta}$, the clock zone post$(\mathcal{Z}, e)$ denotes the set of clock valuations $\mathcal{Z}'$ for which the state $(s', \mathcal{Z}')$ can be reached from the state $(s, \mathcal{Z})$ by letting time elapse and by executing the transition e. The zone $(s', \text{post}(\mathcal{Z}, e))$ describes the discrete successor of the zone $(s, \mathcal{Z})$ under the transition e.



a) Time successor

b) Time predecessor

c) Intersection

d) Clock reset

Figure 5.10: Two clock zones $\mathcal{Z}_1$, $\mathcal{Z}_2$ and examples of operations on them

5.3.5 The Difference Bound Matrices

An important feature of clock zones is that they can be represented by difference bound matrices (DBMs) [Dil90] [BY04], which is the most commonly used data structure for representing clock zones. To get a uniform notation for clock zones in DBMs, it is necessary to introduce a special clock x_0 which is always 0 ($X = X \cup \{x_0\}$). The matrix is indexed by the clocks in X together with the special clock x_0. Furthermore, it is necessary to introduce the domain of *bounds*. A *bound* is an ordered pair $(c,l) \in (\mathbb{Z} \times \{<,\leq\}) \cup \{(\infty,<),(-\infty,<)\}$. The symbols $<$ and $\leq$ are totally ordered (i.e., $<$ is strictly less than $\leq$). The ordering of bounds is defined lexicographically as follows: $(c_1,l_1) \leq (c_2,l_2)$ if either $c_1 < c_2$ or $c_1 = c_2$ and $l_1 = l_2 = \leq$, or $(c_1,l_1) < (c_2,l_2)$ if either $c_1 < c_2$ or $c_1 = c_2$ and $l_1 = <$ and $l_2 = \leq$. The addition is defined as: for every *bound* b, $b + \infty = \infty$. Given two integers c_1 and c_2, $(c_1,\leq)+(c_2,\leq) = (c_1+c_2,\leq)$ and $(c_1,<)+(c_2,<) = (c_1+c_2,<)$ or $(c_1,<)+(c_2,\leq) = (c_1+c_2,<)$.

Definition 44 (Difference Bound Matrix). *A Difference Bound Matrix (DBM) over* *the set of n clocks $\{x_1, x_2, \ldots, x_n\}$ is a $(n+1) \times (n+1)$ square matrix of bounds with rows and columns indexed by $\{x_0, x_1, x_2, \ldots, x_n\}$. The DBM can be represented as follows:*

$$\mathscr{D} = (\boldsymbol{d}_{i,j})_{0 \le i,j \le n}$$

where each $\boldsymbol{d}_{i,j}$ is of the form $(d_{i,j}, \preceq)$, $\preceq \in \{<, \le\}$ and $d_{i,j} \in \mathbb{Z} \cup \{\infty, -\infty\}$ (i.e., a bound). Formally, the semantics of DBM $\mathscr{D}$ is the clock zone:

$$\mathscr{Z} = (x_0 = 0) \ \wedge \bigwedge_{0 \le i \ne j \le n} x_i - x_j \preceq d_{i,j}$$

where x_i is a clock labelling row i, x_j is a clock labelling the column j and the clock x_0 is always equal to 0.

Since the variable x_0 is always equal to 0, it can be used for expressing constraints that only involve a single variable. Thus, $\mathbf{d}_{i,0} = (d_{i,0}, \preceq)$ means $x_i \preceq d_{i,0}$. Similarly, $\mathbf{d}_{0,j} = (d_{0,j}, \preceq)$ means $-x_j \preceq d_{0,j}$ which are rewritten as $x_i - x_0 \preceq d_{i,0}$ and $x_0 - x_j \preceq d_{0,j}$. Further, for each unbounded clock difference $x_i - x_j$, $\mathbf{d}_{i,j} = (\infty, <)$, where the symbol ∞ expresses that no bound exists. Also, for every clock equals itself, $x_i - x_i$, $\mathbf{d}_{i,j} = (0, \le)$.

For instance, consider the clock zone:

$$\mathscr{Z} = [[x_1 \ge 1 \wedge x_2 > 0 \wedge x_2 \le 2 \wedge x_1 - x_2 < 2]]$$

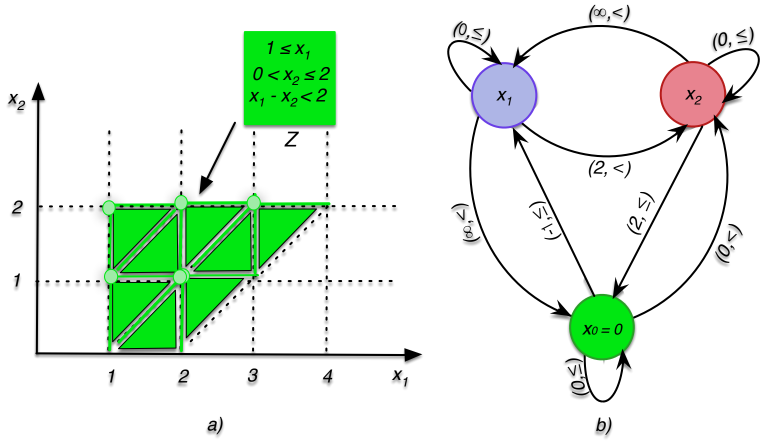


Figure 5.11: a) A clock zone $\mathscr{Z}$ and b) Directed weighted graph

That clock zone can be represented by the matrix $\mathscr{D}$ (DBM), where x_i (resp. x_j) denotes the clock labelling i_{th} row (resp. j_{th} column) of $\mathscr{D}$:

$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \mathscr{D} = \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} (0, \le) & (-1, \le) & (0, <) \\ (\infty, <) & (0, \le) & (2, <) \\ (2, \le) & (\infty, <) & (0, \le) \end{pmatrix} \end{array}$$

that encodes the following inequalities:

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (x_0 - x_0 \le 0) & (x_0 - x_1 \le -1) & (x_0 - x_2 < 0) \\ (x_1 - x_0 < \infty) & (x_1 - x_1 \le 0) & (x_1 - x_2 < 2) \\ (x_2 - x_0 \le 2) & (x_2 - x_1 < \infty) & (x_2 - x_2 \le 0) \end{pmatrix} \end{array}$$

$$\begin{array}{c} \\ x_0 \\ = x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0 \le 0) & (-x_1 \le -1) & (-x_2 < 0) \\ (x_1 < \infty) & (0 \le 0) & (x_1 - x_2 < 2) \\ (x_2 \le 2) & (x_2 - x_1 < \infty) & (0 \le 0) \end{pmatrix} \end{array}$$

In Figure 5.11(a) we can see the representation of the clock zone $\mathscr{Z}$. By a slight abuse of notation, we use the same notations for the DBMs as for clock zones ($\mathscr{Z}$ = $[[\mathscr{D}]]$), writing e.g. $\overrightarrow{\mathscr{Z}}$ as $[[\overrightarrow{\mathscr{D}}]]$ (time successor zone), where $\mathscr{D}$ is a DBM and $\mathscr{Z}$ is a clock zone. It is well known that the representation of a clock zone by a DBM ($\mathscr{Z} = [[\mathscr{D}]]$) is not unique. For instance, if we change $\langle d_{1,0}, \preceq \rangle$ to $(4, <)$ we obtain an alternative DBM for the same clock zone, because $x_1 < 4$ is deduced from $x_1 - x_2 < 2$ and $x_2 \le 2$. To perform such deductions, a clock zone $\mathscr{Z} = [[\mathscr{D}]]$ can be seen as a directed weighted graph [Dil90] [BY04], where each clock corresponds to a node, and each constraint corresponds to a weighted edge, whose weight is represented by $(\mathbf{d}_{i,j})_{0 \le i,j \le n}$. Figure 5.11(b) represents the directed weighted graph for the clock zone $\mathscr{Z} = [[x_1 \ge 1 \wedge x_2 > 0 \wedge x_2 \le 2 \wedge x_1 - x_2 < 2]]$. Then, all deductions can be performed by the composition of the paths.

However, to obtain a unique representation of the zone by its canonical matrix to implement the operations on clock zones (intersection, clock reset, time successor, time predecessor, etc.), it is necessary to obtain a canonical form for DBMs [Dil90] [BY04].

Definition 45 (**Canonical Form of DBMs**). *The canonical form of a DBM can be represented as follows:*

$$\mathscr{D}^c = (\boldsymbol{d}_{i,j})^c_{0 \le i,j \le n},$$

such that, for each $0 \le i, j, k \le n$*, the constraints* $\boldsymbol{d}_{i,j} \le \boldsymbol{d}_{k,j} + \boldsymbol{d}_{i,k}$.

The canonical form of a DBM can be computed by applying the Floyd-Warshall shortest-path algorithm [Dil90].

5.3.5.1 Canonical Form Algorithm

In definition 45, $\mathscr{D}^c[i, j]$ represents the cost of the shortest path in the directed graph of $\mathscr{D}$ from the node with index i to j [Dil90]. Thus, the Floyd-Warshall shortest path algorithm [Dil90] can be used to compute the canonical form representation of a DBM. In short, the principle behind the algorithm is that if there is a negative cost cycle (i.e, a cycle with a cost less than $(0, \le)$) in the directed graph of a given DBM, $\mathscr{D}$, a path of arbitrarily small cost can be obtained by repeating the negative cost cycle ($\mathscr{D}^c[i, j] = (-\infty, <)$ then $x_i - x_j < -\infty$ for each $0 \le i, j \le n$). Therefore, there is a way to decide whether $\mathscr{D}$ has empty clock zones by checking whether a negative

```
1  Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$
2  Output: A DBM $\mathscr{D}^c = (\mathbf{d}_{i,j})_{0\le i,j\le n}$
3  DBM $\mathscr{D}^c$;
4  DBM CanonicalFormOp(DBM $\mathscr{D}$) {
5  int i, j, k;
6     $\mathscr{D}^c$ = $\mathscr{D}$;
7     for(k=0; k<$\mathscr{D}^c$.size; k++){
8       for(i=0; i<$\mathscr{D}^c$.size; i++){
9         for(j=0; j<$\mathscr{D}^c$.size; j++){
10          $\mathscr{D}^c$[i,j] = min($\mathscr{D}^c$[i,j], $\mathscr{D}^c$[i,k] + $\mathscr{D}^c$[k,j]);
11            if (i == j && $\mathscr{D}^c$[i,j] < $(0,\le)$ {
12              $\mathscr{D}^c$[0,0] = $(-\infty,<)$;
13           }
14         }
15       }
16     }
17 return $\mathscr{D}^c$;
18 }
```

Algorithm 5.1: Floyd Warshall Shortest Path Algorithm.

cost cycle occurs during the computation of the shortest path graph using the Floyd-Warshall shortest path algorithm [Dil90]. Algorithm 5.1 describes the Floyd-Warshall shortest path algorithm [Dil90] with a check for emptiness. The algorithm works as follows: at the first iteration, it computes the shortest path among all pairs of nodes present in the directed weighted graph, with the restriction that only the node with index 0 (i.e., x_0) can be visited as an intermediate node. At the second iteration, it computes the shortest path among all pairs of nodes with the restriction that only nodes with index in $(0,1)$ (i.e., x_0 and x_1) can be visited as intermediate nodes. Finally, at iteration n_{th}, it computes the shortest path between all pairs of nodes, using any node in the directed graph of $\mathscr{D}$ as an intermediate node. Note that step 11 checks for negative cost cycles. If a negative-cost cycle is detected, $\mathscr{D}^*[0,0]$ is updated with the value $(-\infty,<)$ to signal that $\mathscr{D}$ represents the empty region, and the algorithm is stopped.

5.3.6 Operations on Zones Using DBMs

In the following, we show some algorithms for the basic operations on the DBM representations of clock zones, which are needed for both the backward and forward analysis techniques. The implementations presented here are taken from [BY04] [BY03], but our presentation differs in some details.

```
Input: Two DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$ and $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\le i,j\le n}$
Output: A DBM $\mathscr{D}'' = (\mathbf{d}_{i,j})_{0\le i,j\le n}$
DBM $\mathscr{D}''$;
DBM IntersectOperator(DBM $\mathscr{D}$, DBM $\mathscr{D}'$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       for(j=0; j<$\mathscr{D}$.size; j++){
         $\mathscr{D}''$[i,j] = min($\mathscr{D}$[i,j], $\mathscr{D}'$[i,j]);
        }
      }
     CanonicalFormOP(DBM $\mathscr{D}''$);
return $\mathscr{D}''$;
}
```

Algorithm 5.2: Intersection Algorithm.

Intersection

Intersection is one of the most common operations performed on DBMs. Given two DBMs $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$ and $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\le i,j\le n}$. Calculate the intersection of these two DBMs $\mathscr{D}'' = (\mathbf{d}''_{i,j})_{0\le i,j\le n}$ so that $[[\mathscr{D}'']] = \mathscr{D} \cap \mathscr{D}'$, consists in taking the lower bounds for each pair of clock differences, i.e. for each $0 \le i,\ j \le n$, $\mathbf{d}''_{i,j} = min(\mathbf{d}_{i,j}, \mathbf{d}'_{i,j})$. However, the intersection operation does not preserve the canonical form of the DBM results, so $\mathscr{D}''$ must be canonicalized after the operation. The algorithm 5.2 describes the intersection operation of two DBMs [Dil90]. The algorithm works as follows: at each iteration, it computes the minimum for each pair of nodes (including other i) present in the two-directional weighted graph.

5.3.6.1 Time Successor

This operation computes the time successor of a non-empty canonical DBM $\mathscr{D}$ (i.e., all clock valuations that can reach $\mathscr{D}$ with any delay). Given a canonical DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$, computing the time successor of a DBM $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\le i,j<n}$, consists in removing in $\mathscr{D}$ all the upper bounds on the values of the clocks, that is, for each $0 < i \le n$, $\mathbf{d}'_{i,0} = (\infty, <)$ and for each $0 < i \le n$, $0 < j \le n$, $\mathbf{d}'_{i,j} = \mathbf{d}_{i,j}$. The Algorithm 5.3 describes the time successor operation of a DBM [Dil90]. The algorithm works as follows: it repeatedly removes the upper bounds of all individual clocks, which is done by replacing all elements in the first column of $\mathscr{D}'$ by $(\infty, <)$.

5.3.6.2 Time Predecessor

This operation computes the time predecessor of a non-empty canonical DBM $\mathscr{D}$ (i.e., all clock valuations can reach $\mathscr{D}$ with any delay). Given a canonical DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$, computing the time predecessor of a DBM $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\le i,j<n}$, consists in removing in $\mathscr{D}$ all the lower bounds on the values of the clocks, that is, for each 0

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$
Output: A DBM $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j\leq n}$
DBM $\mathscr{D}'$;
DBM TimeSuccOperator(DBM $\mathscr{D}$) {
int i, j;
    for(i=0; i<$\mathscr{D}$.size; i++){
        $\mathscr{D}'$[i,0] = $(\infty,<)$;
       }
return $\mathscr{D}'$;
}
```

Algorithm 5.3: Time Successor Algorithm.

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$
Output: A DBM $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j\leq n}$
DBM $\mathscr{D}'$;
DBM TimePredOperator(DBM $\mathscr{D}$) {
int i, j;
    for(i=0; i<$\mathscr{D}$.size; i++){
        $\mathscr{D}'$[i,0] = $(0,\leq)$;
       }
return $\mathscr{D}'$;
}
```

Algorithm 5.4: Time Predecessor Algorithm.

$< i \leq n$, $\mathbf{d}'_{0,i} = (0,\leq)$ and for each $0 < i \leq n$, $0 \leq j \leq n$, $\mathbf{d}'_{i,j} = \mathbf{d}_{i,j}$. The Algorithm 5.4 describes the time predecessor operation of a DBM [Dil90]. The algorithm works as follows: it repeatedly removes the lower bounds of all individual clocks, which is done by replacing all elements in the first column of $\mathscr{D}'$ by $(0,\leq)$.

5.3.6.3 Clock Reset

This operation is used to set clocks of a non-empty canonical DBM $\mathscr{D}$ to zero. Given a canonical DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and given a set of clocks $Y \subseteq X$ to be reset. Computing the clock reset of $Y \subseteq X$ and a DBM $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j<n}$ such that $[[\mathscr{D}']] = \downarrow_Y [[\mathscr{D}]]$, consists in setting to zero all the upper and lower bounds of the values of clocks to be reset ($Y \subseteq X$), that is, for each $0 \leq i \leq n$, such that, $x_i \in Y$ (i.e., x_i denotes the i_{th} clock (row) of the DBM $\mathscr{D}$), $(\mathbf{d}'_{i,0})_{0\leq i<n} = (\mathbf{d}'_{0,i})_{0\leq i<n} = (0,\leq)$ and for each $0 \leq i, j \leq n$, $(\mathbf{d}'_{i,j})_{0\leq i,j<n} = (\mathbf{d}_{i,j})_{0\leq i,j<n}$. The resulting $\mathscr{D}'$ will be in the canonical form [Dil90]. The Algorithm 5.5 describes the reset clock operation of a DBM [Dil90]. The algorithm works as follows: it repeatedly removes the upper bounds of all clocks in Y, which is done by replacing all elements in the first row and column of $\mathscr{D}'$ by $(0,\leq)$, $(\mathbf{d}'_{i,j})_{0\leq i,j<n} = (\mathbf{d}'_{0,j})_{0\leq j<n}$ and $(\mathbf{d}'_{j,i})_{0\leq i,j<n} = (\mathbf{d}'_{j,0})_{0\leq j<n}$ where $(\mathbf{d}'_{0,j})_{0\leq j<n} =$

```
Input: A canonical DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and reset clocks $Y \subseteq X$
Output: A canonical DBM $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$
DBM $\mathscr{D}'$;
DBM ResetOperator(DBM $\mathscr{D}$, Clocks $Y$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       if ($x_i \in Y$){
         for(j=0; i<$\mathscr{D}$.size; j++){
        $\mathscr{D}'$[i,j] = $\mathscr{D}$[0,j];
        $\mathscr{D}'$[j,i] = $\mathscr{D}$[j,0];
        }
       }
     }
return $\mathscr{D}'$;
}
```

Algorithm 5.5: Reset Clock Algorithm.

$(\mathbf{d}'_{j,0})_{0\leq j<n} = (0, \leq)$.

5.3.7 The Extrapolation Abstractions

The zone-based abstractions described in Section 5.3.4 are important approaches to the reachability problem of TA [BY04]. A zone graph captures the behavior of the TA, but such a zone graph is not necessarily finite. Abstraction techniques for zones are used to reduce the number of reachable zones and obtain a finite zone graph [Bou04a]. The idea of these abstraction techniques is to remove from the clock zone those constraints that exceed the bounds used in the guards and invariants. Therefore, the generated clock zone is larger than the original one. In the last 20 years, there have been many interesting advances in abstraction techniques for TA to provide coarser abstractions of TA [Bou04a] [BBFL03] [BBLP06]. The abstraction techniques presented here are taken from [Bou04a] [BBFL03] [BBLP06].

5.3.7.1 Maximal Bounds Abstraction

One of the first works on the zone-based abstraction technique for TA is the so-called *maximal bounds extrapolation* (also known as M-extrapolation or normalization, where M stands for the maximal constant appearing in the constraints of a TA $\mathscr{A}$, i.e., c_x) [DT98]. The basic idea of this abstraction is as follows. Given a TA $\mathscr{A}$, a clock zone $\mathscr{Z}$ is extrapolated by the maximal constant (i.e., the maximal bound c_x or $M(x)$ of a clock $x \in X$) appearing in the guards or invariants with the clock $x \in X$ of $\mathscr{A}$. Formally, it is based on the following equivalence relation. Given two valuations v and v', $\equiv_M$ is the equivalence relation defined by $v \equiv_M v'$ if and only if for all $x \in X$, $v(x) = v'(x)$ or ($v(x) > M(x)$ and $v'(x) > M(x)$). For each clock zone $\mathscr{Z}$, we can describe the extrapolated clock zone by the abstraction function $\mathfrak{a}_M(\mathscr{Z}) = \{v' \mid \exists\, v \in \mathscr{Z},\ v \equiv_M v'\}$.

This abstraction function is complete and sounds [BBFL03] with respect to the reachability problem . But this abstraction is not necessarily used in algorithms because α_M does not preserve the canonical form of clock zones [BBFL03]. The abstraction actually used is an extrapolation denoted by α_{Extra_M} such that for all clock zones $\mathscr{Z}$ we have $\mathscr{Z} \subseteq \alpha_{Extra_M}(\mathscr{Z}) \subseteq \alpha_M(\mathscr{Z})$ and $\alpha_{Extra_M}(\mathscr{Z})$ is a clock zone. Given a clock zone $\mathscr{Z}$, $\alpha_{Extra_M}(\mathscr{Z})$ is computed as follows:

- Removing all upper bounds higher than $M(x)$ with $x \in X$ (i.e., the constraints of the form $x \sim c$ and $x - y \sim c$ where $\sim \in \{<, \leq\}$, $c \in \mathbb{N}$ and $c > M(x)$ are eliminated).
- Lowering all lower bounds higher than $M(x)$ with $x \in X$ down to the $M(x)$ (i.e., the constraints of the form $x \sim c$ and $x - y \sim c$ where $\sim \in \{>, \geq\}$, $c \in \mathbb{N}$ and $c > M(x)$, by $x > M(x)$ and $x - y > M(x)$).

$Extra_M$ is an extrapolation operator on DBMs, it defines the abstraction function, α_{Extra_M}, on zones such that for every clock zone $\mathscr{Z}$, $\alpha_{Extra_M}(\mathscr{Z}) = Extra_M(\mathscr{D})$, where D is the DBM in canonical form which represents the clock zone $\mathscr{Z}$.

Definition 46 (**Extrapolation Operator** $Extra_M$)**.** *Let $\mathscr{Z}$ be a clock zone represented by a DBM in a canonical form $\mathscr{D} = (\boldsymbol{d}_{i,j})_{0\leq i,j\leq n}$ (where $\boldsymbol{d}_{i,j} = (d_{i,j}, \preceq_{i,j})_{0\leq i,j\leq n}$). For each clock $x_i \in \mathscr{A}$, $M(x_i)$ is the maximal bound that appears in the guard and invariants of $\mathscr{A}$. The extrapolation operator $\mathscr{D}' = Extra_M(\mathscr{D})$ with $\mathscr{D}' = (\boldsymbol{d}'_{i,j})_{0\leq i,j\leq n}$ is defined as follows:*

$$\boldsymbol{d}'_{i,j} = \begin{cases} (\infty, <) & if \quad d_{i,j} > M(x_i) \\ (-M(x_j), <) & if \quad -d_{i,j} > M(x_j) \\ (d_{i,j}, \prec_{i,j}) & \qquad otherwise \end{cases}$$

Algorithm 5.6 describes the $Extra_M$ operation on a DBM [BBFL03]. The algorithm works as follows: it repeatedly removes all upper bounds higher than $M(x_i)$ with $x_i \in X$ and lowers all lower bounds higher than $M(x_i)$ with $x_i \in X$ down to the $M(x_i)$. Note that the $Extra_M$ operator does not preserve the canonical form of the DBM.

5.3.7.2 Lower and Upper Maximal Bounds Abstraction

An extension to the maximal bounds abstraction was introduced in [BBLP04]. This new extension is based on the fact that the maximal lower and upper bounds to which clocks of a TA are compared often differ. The basic idea is as follows. Given a TA $\mathscr{A}$, a clock zone $\mathscr{Z}$ is extrapolated (i.e., bounded) by two bounds (i.e., the maximal lower bound $L(x)$ and maximal upper bound $U(x)$ of a clock $x \in X$) appearing in the guards or invariants with the clock $x \in X$ of $\mathscr{A}$. Naturally, it holds that $M(x) = max(L(x), U(x))$. Formally, this new abstraction is based on the following simulation relation. Given two valuations v and v', $\preceq_{LU}$ is the relation defined by $v \preceq_{LU} v'$ if and only if for all $x \in X$, $v(x) = v'(x)$ or $L(x) < v(x) < v'(x)$ or $U(x) < v'(x) < v(x)$. For every clock zone $\mathscr{Z}$, we can describe the extrapolated clock zone by the abstraction function $\alpha_{LU}(\mathscr{Z}) = \{v' \mid \exists\, v \in \mathscr{Z},\ v \preceq_{LU} v'\}$. This abstraction function is sound and complete with regards to reachability problem [BBLP04] [BBLP06].

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$ and $M(x_i)$ the maximal bound, where $x_i \in$
        $X$.
Output: A DBM $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\le i,j\le n}$.
DBM $\mathscr{D}'$;
DBM ExtraMOperator(DBM $\mathscr{D}$, List const $M$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       for(j=0; j<$\mathscr{D}$.size; j++){
         if (i $\neq$ j && $\mathscr{D}$[i,j] > ($M(x_i)$, <)) {
            $\mathscr{D}'$[i,j] = ($\infty$,<);
        }
        else if($\mathscr{D}$[i,j] < ($-M(x_j)$, <)) {
           $\mathscr{D}'$[i,j] = ($-M(x_j)$, <);
       }
      }
     }
    CanonicalFormOp(DBM $\mathscr{D}'$);
return $\mathscr{D}'$;
}
```

Algorithm 5.6: $Extra_M$ Algorithm

But this abstraction is not used in algorithms because $\mathfrak{a}_M$ does not preserve clock zones [BBLP04] [BBLP06]. The abstraction actually used is an extrapolation denoted by $\mathfrak{a}_{Extra_{LU}}$ such that for all clock zones $\mathcal{Z}$ we have $\mathcal{Z} \subseteq \mathfrak{a}_{Extra_{LU}}(\mathcal{Z}) \subseteq \mathfrak{a}_{LU}(\mathcal{Z})$ and $\mathfrak{a}_{Extra_{LU}}(\mathcal{Z})$ is a clock zone. Given a clock zone $\mathcal{Z}$, $\mathfrak{a}_{Extra_{LU}}(\mathcal{Z})$ is computed as follows:

- Removing all lower bounds higher than $L(x)$ with $x \in X$ (i.e., the constraints of the form $x \sim c$ and $x - y \sim c$ where $\sim \in \{<, \le\}$, $c \in \mathbb{N}$ and $c > L(x)$ are eliminated).
- Lowering all upper bounds higher than $U(x)$ with $x \in X$ down to the $U(x)$ (i.e., the constraints of the form $x \sim c$ and $x - y \sim c$ where $\sim \in \{>, \ge\}$, $c \in \mathbb{N}$ and $c > U(x)$ are replaced by $x - y > U(x)$).

$Extra_{LU}$ is an extrapolation operator on DBMs, it defines the abstraction function, $\mathfrak{a}_{Extra_{LU}}$, on zones such that for every clock zone $\mathcal{Z}$, $\mathfrak{a}_{Extra_{LU}}(\mathcal{Z}) = Extra_{LU}(\mathscr{D})$, where D is the DBM in canonical form which represents the clock zone $\mathcal{Z}$.

Definition 47 (**Extrapolation Operator** $Extra_{LU}$)**.** *Let $\mathcal{Z}$ be a clock zone represented by a DBM in a canonical form $\mathscr{D} = (\boldsymbol{d}_{i,j})_{0\le i,j\le n}$ (where $\boldsymbol{d}_{i,j} = (d_{i,j}, \preceq_{i,j})_{0\le i,j\le n}$). For each clock $x_i \in \mathscr{A}$, $L(x_i)$ is the maximal lower bound and $U(x_i)$ is the maximal upper bound that appears in a guard or invariants of $\mathscr{A}$. The extrapolation operator $\mathscr{D}' = Extra_{LU}(\mathscr{D})$ with $\mathscr{D}' = (\boldsymbol{d}'_{i,j})_{0\le i,j\le n}$ is defined as follows:*

$$\boldsymbol{d}'_{i,j} = \begin{cases} (\infty, <) & if \quad d_{i,j} > L(x_i) \\ (-U(x_j), <) & if \quad -d_{i,j} > U(x_j) \\ (d_{i,j}, \prec_{i,j}) & \quad otherwise \end{cases}$$

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$, $L(x_i)$ the maximal lower bound (and
    $U(x_i)$ the maximal upper bound), where $x_{i_{0\le i<n}} \in X$.
Output: A DBM $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\le i,j\le n}$.
DBM $\mathscr{D}'$;
DBM ExtraLUOperator(DBM $\mathscr{D}$, List const $L$, List const $U$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       for(j=0; j<$\mathscr{D}$.size; j++) {
         if (i $\neq$ j && $\mathscr{D}$[i,j] > ($L(x_i$, <))) {
            $\mathscr{D}'$[i,j] = $(\infty, <)$;
        }
        else if($\mathscr{D}$[i,j] < ($-U(x_j)$, <)) {
           $\mathscr{D}'$[i,j] = ($-U(x_j)$, <);
        }
      }
     }
    CanonicalFormOp(DBM $\mathscr{D}'$);
return $\mathscr{D}'$;
}
```

Algorithm 5.7: $Extra_{LU}$ Algorithm

Algorithm 5.7 describes the $Extra_{LU}$ operation on a DBM [BBFL03]. The algorithm works as follows: it repeatedly removes all upper bounds higher than $U(x_i)$ with $x_i \in X$ and lowers all lower bounds higher than $L(x_i)$ with $x_i \in X$ down to the $L(x_i)$. Note that the $Extra_{LU}$ operator does not preserve the canonical form of the DBM.

5.3.8 Reachability Algoritm

Now, to solve the reachability problem for a considered zone automata, the following general algorithm 5.8 is used [BY04]: The idea is to make a forward exploration of the zone graph in some search order (e.g., breadth-first or depth-first search), computing the set of all zones reachable from the initial configuration $(s_0, \mathcal{Z}_0)$). It uses two sets of zones (Passed and Waiting) to explore the zone graph starting from $(s_0, \mathcal{Z}_0)$). At each step, if a new zone is obtained from $Wait$, it is checked if the new zone is in $Passed$, and if not, it is added to $Passed$ and its successors to $Wait$. All operations on used zones are implemented using DBMs.

5.3.9 Timed Bisimulation

The state explosion problem [PT87] is one of the most serious problems encountered by model checking (notably on RTS, the state space can be huge). The notion of bisimulation relation [Mil89] has been used to mitigate the state explosion problem and strong timed bisimulation has been used to reason about the behavior of RTS.

```
Input: A TA $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F)$ and an initial zone $(s_0, \mathcal{Z}_0)$.
Output: All zone states reachable from $(s_0, \mathcal{Z}_0)$.
//$s \in S$ is a location of $\mathscr{A}$, $\mathcal{Z}_0$, $\mathcal{Z}$ and $\mathcal{Z}_f$ are DBM.
Zones $Passed$;
Zones $Wait$;
Zones ReachGeneralAlgo(TA $\mathscr{A}$) {
Passed = $\emptyset$;
Waiting = $(s_0, Extra^{+}_{LU}(\mathcal{Z}_0))$; //for all $x \in X$ and $v \in \mathcal{Z}_0$, $v(x) = 0$
while ($Wait \neq \emptyset$){
 Choose and Remove a zone $(s, \mathcal{Z})$ from $Wait$
      if foreach $(s_f, \mathcal{Z}_f) \in Passed$, $s \neq s_f$ or $\mathcal{Z} \not\subseteq \mathcal{Z}_f$ {
         add $(s, Extra^{+}_{LU}(\mathcal{Z})$ to $Passed$
         foreach $e = (s, a, \phi, Y, s') \in \rightarrow_{ta}$ {
           if(post($\mathcal{Z}$,$e$) $\neq \emptyset$)
            {
            add $(s', Extra^{+}_{LU}$(post($\mathcal{Z}$,$e$))) to $Wait$
            }
         }
       }
}
return $Passed$;
}
```

Algorithm 5.8: Algorithm for reachability problem using zones.

Definition 48 (**Strong Timed Bisimulation** [Cer93])**.** *Let $\mathscr{D}_1$ and $\mathscr{D}_2$ be two TLTS over alphabet Σ. Let $Q_{\mathscr{D}_1}$ (resp., $Q_{\mathscr{D}_2}$) be the set of states of $\mathscr{D}_1$ (resp., $\mathscr{D}_2$). A strong timed bisimulation over $\mathscr{D}_1$ and $\mathscr{D}_2$ is a binary relation $\mathscr{R} \subseteq Q_{\mathscr{D}_1} \times Q_{\mathscr{D}_2}$ such that the following holds: if $q_{\mathscr{D}_1} \mathscr{R} q_{\mathscr{D}_2}$ then:*

(i) Discrete transition: For every $q_{\mathscr{D}_1} \xrightarrow{a}_{\mathscr{D}_1} q'_{\mathscr{D}_1}$ with $a \in \Sigma$, there exists a matching transition $q_{\mathscr{D}_2} \xrightarrow{a}_{\mathscr{D}_2} q'_{\mathscr{D}_2}$ such that $q'_{\mathscr{D}_1} \mathscr{R} q'_{\mathscr{D}_2}$ and vice versa,

(ii) Delay transition: For every $q_{\mathscr{D}_1} \xrightarrow{d}_{\mathscr{D}_1} q'_{\mathscr{D}_1}$ with $d \in \mathbb{R}_{\geq 0}$, there exists a matching transition $q_{\mathscr{D}_2} \xrightarrow{d}_{\mathscr{D}_2} q'_{\mathscr{D}_2}$ such that $q'_{\mathscr{D}_1} \mathscr{R} q'_{\mathscr{D}_2}$ and vice versa.

Two states $q_{\mathscr{D}_1}$ and $q_{\mathscr{D}_2}$ are timed bisimilar, written $q_{\mathscr{D}_1} \sim q_{\mathscr{D}_2}$, iff there is a timed bisimulation $\mathscr{R}$ such that $q_{\mathscr{D}_1} \mathscr{R} q_{\mathscr{D}_2}$. $\mathscr{D}_1$ and $\mathscr{D}_2$ are timed bisimilar, written $\mathscr{D}_1 \sim \mathscr{D}_2$, if there exists a timed bisimulation relation $\mathscr{R}$ over $\mathscr{D}_1$ and $\mathscr{D}_2$ containing the pair of initial states. Furthermore, for all $q_{\mathscr{D}_1} \mathscr{R} q_{\mathscr{D}_2}$, if $q_{\mathscr{D}_1} \in Q^F_{\mathscr{D}_1}$ then $q_{\mathscr{D}_2} \in Q^F_{\mathscr{D}_2}$.

5.3.10 Partition Refinement Algorithm

Based on the fact that the reachability algorithm induces a finite zone graph (see section 5.3.4), the strong bisimulation can be computed using the well-known partition refinement algorithm [PT87]. Essentially, the algorithm divides the state space Q into blocks (i.e., pairwise disjoint sets of states). Starting with an initial partition Π

= Π_0 of the state space Q, where all states with the same label form a partition, the algorithm successively refines these partitions until a stable partition Π is reached. A partition is stable if and only if elements of Π become equivalence classes of some equivalence relation, which is a bisimulation.

Remarkably, a bisimulation can induce a pre-stable partition. The algorithm works as follows: given a partition of states Π and blocks B_1, $B_2 \in \Pi$, B_1 is pre-stable concerning B_2 if $B_1 \subseteq pred(B_2)$ or $B_1 \cap pred(B_2) = \emptyset$, where $pred(B)$ is the set of direct predecessors of all the states in B. If B_1 is not stable with respect to B_2, then B_1 can further be partitioned into two sub-blocks $B_1 \cap pred(B_2)$ and $B_1 \setminus pred(B_2)$. In this case, B_2 is a splitter of B_1. Π is pre-stable if all its blocks are pairwise pre-stable.

In [TY01] such an adaptation is given for TA to compute the time-abstracting bisimulation [Tri98]. Given a timed automaton $\mathscr{A}$, the idea is to start from an initial partition Π_0 respecting the invariants and guards of $\mathscr{A}$ and to refine it successively so that each block (i.e. the blocks are represented by DBM) is pre-stable with respect to all its time successors (delay transitions) or with respect to all its action successors (discrete transitions) for a given action [TY01]. Delay transitions must guarantee that the time successor continuously traverses backward diagonal time successor clock zones (e.g., any elapsing time cannot traverse the clock zones from 1 to 3 without traversing 2) and alternates with a discrete transition. Figure 5.12 shows an example of a delay transition traversing clock zones 1 through 3 (i.e., clocks x and y are perfectly synchronous). Extensions of the partitioning refinement algorithm for real-time systems are also described in [HNSY94], [DY96], and [Tri98].

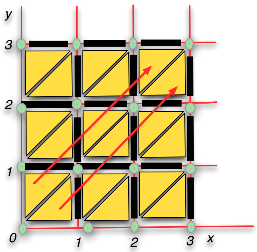


Figure 5.12: A delay transition traversing the range 0 to 3

5.4 Event Clock Automata

Event clock automata (ECA) [AFH94] are a strict subclass of TA, in which clock resets are not arbitrary (i.e., each action a is associated with a clock x_a that is reset exactly when the action a occurs). ECA are closed under union and intersection, and they can be determinized (unlike TA). A ECA over Σ= $\{a, b\}$ uses two implicit clocks x_a, x_b and y_a, y_b for each $a, b \in \Sigma$. Along TT (or TIS) the clock x_a measures the time since the last occurrence (event recording) of the symbol a (x_b for the symbol b) and y_a measures the time until the next occurrence (event predicting) of the symbol a (y_b for the symbol b). The set of all (event recording) and (event predicting) clocks is denoted by $X = \{x_a, x_b, y_a, y_b \mid a, b \in \Sigma\}$. However, ECA is strictly less expressive than TA [AFH94]. Event Recording Automata (ERA) [AFH94] is a subclass of ECA that is expressive enough to model TLTS.

5.4.1 Event Clock Automata with Pointwise Semantics

Here, we define formally the notion of event clocks, event clock constraints, event clock valuations, and ECA with pointwise semantics.

5.4.1.1 Event Clocks

Let $\Sigma = 2^{\mathbb{P}}$ be a finite alphabet of symbols. Let $X = \{R, P\}$ be a finite set of event clocks associated to Σ where $R = \{x_p \mid p \in \Sigma\}$, is the set of event recording clocks and $P = \{y_p \mid p \in \Sigma\}$, is the set of event predicting clocks. In what follows, we note z any recording and predicting clock of X.

5.4.1.2 Event Clock Constraints

Let X be a finite set of event clocks associated to Σ. Let $\Phi(X)$ be a set of clock constraints over X. A clock constraint $\phi \in \Phi(X)$ can be defined by the following grammar:

$$\phi := true \mid z \sim c \mid \phi_1 \wedge \phi_2$$

5.4.1.3 Event Clock Valuations

Let X be a finite set of event clocks. A clock valuation function, $\nu : X \to (\mathbb{R}_{\geq 0} \cup \{\perp\})$ assigning to each event clock $x \in X$ a value in $(\mathbb{R}_{\geq 0} \cup \{\perp\})$, where $\perp$ denotes the undefined value. We note $(\mathbb{R}^X_{\geq 0} \cup \{\perp\})$ the set of all clock valuations. Given an TT ρ and position $i \in \mathbb{N}_{\geq 0}$, a clock valuation $\nu(\rho, i) \in (\mathbb{R}^X_{\geq 0} \cup \{\perp\})$ over X, specifies the values of the event clocks in X at position i in ρ.

Given an TT ρ, the valuation of the event recording clock variable x_p along a ρ at position i is:

$$\nu(\rho, i, x_p) = \begin{cases} t_i - t_j & \textit{if } \exists j < i \textit{ such that } p = \sigma_j, \textit{ and} \\ & \forall k, j < k < i \textit{ such that } p \neq \sigma_k, \\ & \\ \perp & \textit{else} \end{cases}$$

Given a TT ρ, the valuation of the event predicting clock variable y_p along a ρ at position i is:

$$\nu(\rho, i, y_p) = \begin{cases} t_j - t_i & \textit{if } \exists j > i \textit{ such that } p = \sigma_j, \textit{ and} \\ & \forall k, i < k < j \textit{ such that } p \neq \sigma_k, \\ & \\ \perp & \textit{else} \end{cases}$$

Given a clock constraint $\phi \in \Phi(X)$, a TT ρ, we say that (ρ, i) satisfies ϕ at a position i, denoted by $(\rho, i) \models \phi$ and defined formally as follows:

$$(\rho, i) \models x \sim c \Longleftrightarrow \nu(\rho, i, z) \sim c$$

$$(\rho, i) \models \phi_1 \wedge \phi_2 \Longleftrightarrow \nu(\rho, i, z) \models \phi_1 \wedge \nu(\rho, i, z) \models \phi_2$$

$$(\rho, i) \models true \Longleftrightarrow true$$

Definition 49. *An ECA in the pointwise semantics is a tuple $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{eca}, F)$, such that:*

(i) S is a finite set of locations,
(ii) $s_0 \in S$ is the initial location,
(iii) Σ is a finite alphabet. We take $\Sigma = 2^{\mathbb{P}}$, where $\mathbb{P}$ is a finite set of propositional symbols,
(iv) X is a finite set of clocks,
(v) $\rightarrow_{eca} \subseteq S \times \Sigma \times \Phi(X) \times S$ is a transition relation,
(vi) $F = \{F_1, F_2, \ldots\}$ is a set of accepting locations (Büchi acceptance condition).

An accepted run θ of $\mathscr{A}$ on a TT ρ is a infinite sequence $\theta = s_0 \xrightarrow{e_0} s_1 \xrightarrow{e_1} \ldots s_n \xrightarrow{e_n} \ldots$ where $s_i \in S$, $s_0 \in S$, $e_i = (s_i, p, \phi_i, s_{i+1}) \in \rightarrow_{eca}$, $(\rho, i) \models \phi_i$ and there exists infinitely many positions j such that $s_j \in F_i$ for $i \geq 1$.

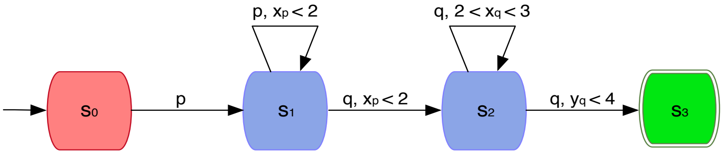


Figure 5.13: Event Clock Automata

Example 14. *Let us consider the ECA of Figure 5.13. This ECA contains 4 locations, s_0 is a start location. All edges that are not labeled by clock constraints have, by default, the trivial clock constraint true. The clock constraint $x_p < 2$ that is associated with the loop from s_1 to itself ensures that each p occurs within 2 time units of the preceding p. The clock constraint $x_p < 2$ that is associated with the edge from s_1 to s_2 ensures that q occurs within 2 time units of the preceding q. A similar mechanism for checking the value of x_q while reading q ensures that the time difference between each q and the subsequent q is always greater than 2 and less than 3. The clock constraints $y_q < 4$ that is associated with the edge from s_2 to s_3 ensures that q must occurs within 4 time units.*

Theorem 24. *[AFH94] The ECA with pointwise semantics are closed under all boolean operations.*

Theorem 25. *[AFH94] The emptiness problem for ECA with pointwise semantics is decidable and PSPACE-Complete.*

Theorem 26. *[AFH94] The problem of universality for ECA with pointwise semantics is decidable and PSPACE-Complete.*

5.4.2 Event Clock Automata with Continuous Semantics

Now, we define formally the notion of ECA with continuous semantics. Here, we use, ECA with some changes regarding the definition done in [RS97] (and 5.4.1). The main changes are: (1) labeled each location in ECA with atomic propositional $\Sigma = 2^{\mathbb{P}}$, (2) labeled each location in ECA with event recording and event predicting invariants, (3) labeled the constraints with event recording and event predicting clocks and (3) worked with TIS instead of TT.

5.4.2.1 Event Clocks

The definition of event clocks associated to $\Sigma = 2^{\mathbb{P}}$ is as in pointwise semantics 5.4.1 (event recording and event predicting clocks denoted by $X = \{x_p, y_p \mid p \in \mathbb{P}\}$). Thus, the value of a clock is the time elapsed since its last reset. However, when we use continuous time, there is not always a last reset, e.g. when the reset holds in an open interval. For this case, we will use non-standard clock values of the form v^+, intuitively meaning that the clock was reset v units before. The set of non-standard real numbers, noted $\mathbb{R}^+_{\geq 0}$, is the set of $\{v, v^+ \mid v \in \mathbb{R}_{\geq 0}\}$, ordered by $<_{ns}$ as the following: $v_1 <_{ns} v_2^+$ iff $v_1 \leq v_2$. The addition is commutative, and $v_1^+ + v_2 = (v_1 + v_2)^+$. In what follows, we note z any recording and predicting clock of X.

5.4.2.2 Event Clock Invariants

Let X be a finite set of event clocks ranging over $\mathbb{R}_{\geq 0}$. Let $\Delta(X)$ be a set of event clock invariants. An event clock invariant $\varphi \in \Delta(X)$ is a clock constraints that can be defined by the following form:

$$\varphi := true \mid z < c \mid z \leq c \mid \varphi_1 \wedge \varphi_2$$

where $z \in X$, $c \in \mathbb{N}$.

5.4.2.3 Event Clock Constraints

The definition of event clock constraints is as in pointwise semantics 5.4.1.

5.4.2.4 Event Clock Valuations

Let X be a finite set of event clocks. A clock valuation function, $\nu : X \rightarrow (\mathbb{R}_{\geq 0} \cup \{\perp\})$ assigning to each event clock $x \in X$ a value in $(\mathbb{R}_{\geq 0} \cup \{\perp\})$, where $\perp$ denotes the undefined value. We note $(\mathbb{R}^X_{\geq 0} \cup \{\perp\})$ the set of all clock valuations. Given an TIS ρ and time $t \in \mathbb{R}_{\geq 0}$, a clock valuation $\nu(\rho, t) \in (\mathbb{R}^X_{\geq 0} \cup \{\perp\})$ over X, specifies the values of event clocks in X at time t in ρ. Given an TIS ρ, the valuation of the event recording clock variable x_p along a ρ at time t is:

$$\nu(\rho,t,x_p)=\begin{cases} t-r & if\ \ p\in\rho(t-r),\ r>0,\ \ and\\ & \forall r', 0<r'<r\ \ such\,that\ \ p\notin\rho(t-r'),\\ \\ (t-r)^+ & if\ \ \forall r'>r,\ \exists r'', r<r''<r'\ \ such\,that\ \ p\in\rho(t-r''),\ \ and\\ & \forall r', 0<r'\le r\ \ such\,that\ \ p\notin\rho(t-r')\\ \\ \perp & if\ \ \forall r, 0<r\le t\ \ such\,that\ \ p\notin\rho(t-r) \end{cases}$$

Given a TIS ρ, the valuation of the event recording clock variable y_p along a ρ at time t is:

$$\nu(\rho,t,y_p)=\begin{cases} t+r & if\ \ p\in\rho(t+r),\ r>0,\ \ and\\ & \forall r', 0<r'<r\ \ such\,that\ \ p\notin\rho(t+r'),\\ \\ (t+r)^+ & if\ \ \forall r'>r,\ \exists r'', r<r''<r'\ \ such\,that\ \ p\in\rho(t+r''),\ \ and\\ & \forall r', 0<r'\le r\ \ such\,that\ \ p\notin\rho(t+r')\\ \\ \perp & if\ \ \forall r, 0<r\le t\ \ suchthat\ \ p\notin\rho(t+r) \end{cases}$$

Definition 50. *An ECA in the continuous semantics is a tuple $\mathscr{A}=(S,s_0,\Sigma,X,\gamma,Inv,\rightarrow_{eca},F)$, such that:*

(i) S is a finite set of locations,
(ii) $s_0\in S$ is the initial location,
(iii) Σ is a finite alphabet. We take $\Sigma=2^{\mathbb{P}}$, where $\mathbb{P}$ is a finite set of propositional symbols,
(iv) X is a finite set of clocks,
(v) $\gamma: S\rightarrow\Sigma$ is a function which labels each location $s\in S$ with the set of propositions that are true in that location,
(vi) $Inv: S\rightarrow\Phi(C)$ is a labeling function assigning a clock constraint to each location $s\in S$, where $\Phi(C)$ denote the set of clock constraints and C denote the set of event clocks,
(vii) $\rightarrow_{eca}\subseteq S\times S$ is a transition relation,
(viii) $F\subseteq S$ is a set of accepting locations.

An ECA $\mathscr{A}$ accepts an TIS ρ, if there exist an accepted infinite $run\ \theta=(s,I)$ such that the following conditions hold: the run starts in a starting location $s_0\in S$, the run θ consists of an infinite sequence of locations s and an infinite sequence of intervals I that cover $[0,\infty)$, where $(s_0,I_0)\rightarrow(s_1,I_1)\rightarrow(s_2,I_2)\cdots(s_{n-1},I_{n-1})\rightarrow(s_n,I_n)\rightarrow\cdots$ with $s_i\in S$, and $I_i\in I$, for all $i>0$ there is a transition in $\rightarrow_{eca}$ of the form (s_{i-1},s_i), such that $\gamma(\theta(t))=\rho(t)$ and $\nu(\rho,t)\models Inv(\theta(t))$ for all $t\in[0,\infty)$, and if there exist infinitely many $i\geq 0$, such that $s_i\in F$ is a Büchi accepting state that occurs infinitely often in θ. The timed language $\mathscr{L}(\mathscr{A})$ defined by the ECA $\mathscr{A}$ consists of all TIS ρ that $\mathscr{A}$ accepts.

Example 15. *Let us consider the railroad crossing gate problem that is taken from [HL94]. Let us assume that the train signals its approaching of the gate and imposes*

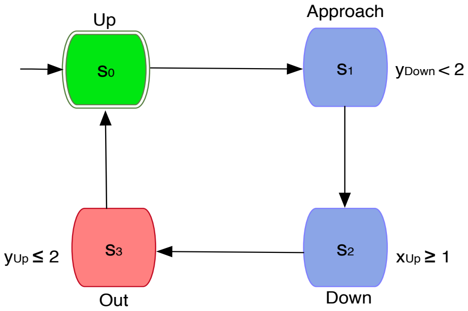


Figure 5.14: Labelled Event Clock Automata from [HL94]

that the gate down before 2 time units. Also, it is assumed that the gate takes at least 1 time unit to go from up to down. The ECA for the railroad crossing gate is considered in Figure 5.14. On approaching the gate, the clocks are set to zero, and in the location s_0 the proposition is Up. When the train has signaled its approaching, the automaton evolves to the location s_1, which is annotated by the constraint $y_{Down} < 2$ (gate down before 2 time units). In the location s_2, which is annotated by the proposition $Down$ and the constraint $x_{Up} \geq 1$ imposing that the gate takes at least 1 time unit to go from location Up to location $Down$. When the train has signaled its rising (out) the automaton evolves to the location s_3, which is annotated by the constraint $y_{Down} \leq 2$, imposing that the gate takes at most 2 time unit to go up.

Theorem 27. *[AFH94] The ECA with continuous semantics are closed under all boolean operations.*

Theorem 28. *[AFH94] The emptiness problem for ECA with continuous semantics is decidable and PSPACE-Complete.*

Theorem 29. *[AFH94] The problem of universality for ECA with continuous semantics is decidable and PSPACE-Complete.*

5.5 Recursive Event Clocks Automata

The event clock values are deterministic, and thus ECA is determinizable, which makes language inclusion decidable and thus allows refinement based development [AFH94]. However, the expressiveness of ECA is rather weak. Therefore, [Ras99, HRS98] introduced the notion of recursive event. In a recursive event model, the reset of an event clock $x_{\mathscr{B}}$ is decided by a lower-level automaton (or formula) $\mathscr{B}$. Thus, when $\mathscr{B}$ visits monitored places (or transitions), it resets $x_{\mathscr{B}}$. Symmetrically, prediction clocks of the form $y_{\mathscr{B}}$ measure the time until $\mathscr{B}$ can next visit one of its monitored locations (or transitions). So no automaton can reset its clocks. In particular, an automaton of level 0 has no sub-automatons, so no clock. RECA can be determinized and thus complemented: They are fully decidable [Ras99, HRS98].

They are quite expressive, since they can express the logic MITL. [AFH96], but less expressive than TA (otherwise we would lose full decidability).

5.5.1 Event Clocks

Let $\mathscr{B}$ be a lower-level automaton. Let $X = \{R, P\}$ be a finite set of event clocks associated to the lower-level automaton $\mathscr{B}$ where $R = \{x_{\mathscr{B}} \mid \mathscr{B}$ is a lower-level automaton$\}$, is the set of event recording clocks and $P = \{y_{\mathscr{B}} \mid \mathscr{B}$ is a lower-level automaton$\}$, is the set of event predicting clocks. Here, we will use non-standard clock values, as in continuous semantics 5.4.2. In what follows, we note $z_{\mathscr{B}}$ any recording and predicting clock of X associated to the lower-level automaton $\mathscr{B}$.

5.5.2 Event Clock Invariants

Let $\mathscr{B}$ be a lower-level automaton. Let Xbe a finite set of event clocks ranging over $\mathbb{R}_{\geq 0}$. Let $\Delta(X)$ be a set of event clock invariants. An event clock invariant $\varphi \in \Delta(X)$ is a clock constraints that can be defined by the following form:

$$\varphi := true \mid z_{\mathscr{B}} < c \mid z_{\mathscr{B}} \leq c \mid \varphi_1 \wedge \varphi_2$$

where $z_{\mathscr{B}} \in X$, $c \in \mathbb{N}$.

5.5.3 Event Clock Constraints

Let $\mathscr{B}$ be a lower-level automaton. Let X be a finite set of event clocks associated to Σ. Let $\Phi(X)$ be a set of clock constraints over X. An event clock constraint $\phi \in \Phi(X)$ can be defined by the following form:

$$\phi := true \mid z_{\mathscr{B}} \sim c \mid \phi_1 \wedge \phi_2$$

where $z_{\mathscr{B}} \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, >, \leq, \geq, =\}$.

5.5.4 Event Clock Valuations

Let $\mathscr{B}$ be a lower-level automaton. Let X be a finite set of event clocks. A clock valuation function, $v : X \rightarrow (\mathbb{R}_{\geq 0} \cup \{\bot\})$ assigning to each event clock $x \in X$a value in $(\mathbb{R}_{\geq 0} \cup \{\bot\})$, where $\bot$ denotes the undefined value. We note $(\mathbb{R}^X_{\geq 0} \cup \{\bot\})$ the set of all clock valuations. Given an TIS ρ and time $t \in \mathbb{R}_{\geq 0}$, a clock valuation $v(\rho, t) \in (\mathbb{R}^X_{\geq 0} \cup \{\bot\})$ over X, specifies the values of event clocks in X at time t in ρ.

Given a TIS ρ and lower-level automaton $\mathscr{B}$, the valuation of the event recording clock variable $x_{\mathscr{B}}$ along a ρ at time t is:

$$v(\rho, t, x_{\mathscr{B}}) = \begin{cases} t - r & \text{if } r = \max\{t' < t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B})\} \text{ exists} \\ (t - r)^+ & \text{else, if } r = \sup\{t' < t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B})\} \text{ exists} \\ \bot & \text{else} \end{cases}$$

Given an TIS ρ and lower-level automaton $\mathscr{B}$, the valuation of the event recording clock variable y_p along a ρ at time t is:

$$v(\rho, t, y_{\mathscr{B}}) = \begin{cases} l-t & \text{if } l = \min\{t' > t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B})\} \text{ exists} \\ (l-t)^+ & \text{else, if } l = \inf\{t' > t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B})\} \text{ exists} \\ \perp & \text{else} \end{cases}$$

Definition 51. *A RECA $\mathscr{A}$ of level $l \in \mathbb{N}$ is a tuple $\mathscr{A} = (S, S_0, \Sigma, X, \gamma, Inv, \rightarrow_{reca}, M, F)$, such that:*

(i) S is a finite set of locations,

(ii) $S_0 \subseteq S$ are the initial locations,

(iii) Σ is a finite alphabet. We take $\Sigma = 2^{\mathbb{P}}$, where $\mathbb{P}$ is a finite set of propositional symbols,

(iv) X is a finite set of clocks, of the form $x_{\mathscr{B}}$ or $y_{\mathscr{B}}$, with $\mathscr{B}$ a lower-level RECA,

(v) γ: $(S \cup \rightarrow_{reca}) \rightarrow \Sigma$ is a labelling function,

(vi) Inv: $(S \cup \rightarrow_{reca}) \rightarrow \Phi(C)$ gives the guard or invariant,

(vii) $\rightarrow_{reca} \subseteq S \times S$ are the transitions.

(viii) $M \subseteq (S \cup \rightarrow_{reca})$ is the set of monitored locansitions or transitions: when the automaton visits them, it resets its two associated clocks $x_{\mathscr{B}}, y_{\mathscr{B}}$.

(ix) F is an acceptance condition.

Definition 52. *A run θ of a RECA $\mathscr{A}$ is an TIS of alternating transitions and locations $(\zeta_0, s_1, \zeta_1, s_2, \ldots, I)$, such that:*

(i) The run starts from an initial location: $\zeta_0 \in S_0 \times S$,

(ii) The run follows discrete transitions: $\zeta_i = (s_i, s_{i+1}) \in \rightarrow_{reca}$,

(iii) The clock constraints (invariant or guard) are satisfied by the valuation of the clocks (defined below): $\forall\, t \in \mathbb{R}_{\geq 0}, v(\rho, t) \models Inv(\theta(t))$.

(iv) It satisfies the acceptance condition.

Definition 53. *The TIS ρ of a run θ, noted $\gamma(\rho)$, is the pair $(\gamma(s), I)$.*

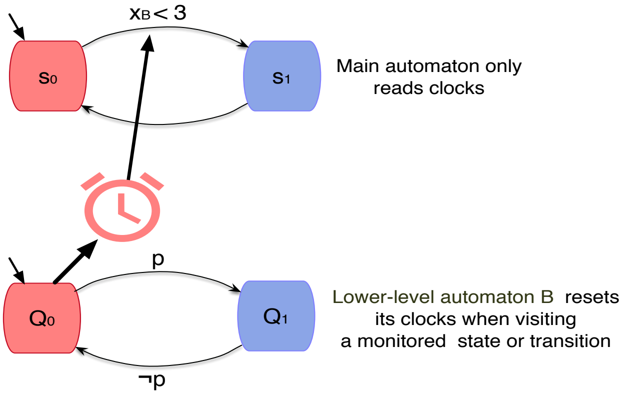
Definition 54. *$\mathscr{A}$ accepts an TIS ρ at t, if there is a run θ for ρ that visits a monitored location at t.*

Example 16. *Let us consider the RECA of the Figure 5.15. This RECA contains two automata, the main automaton contains two locations, s_0 is a start location. The constraints $x_{\mathscr{B}} < 3$ decorating the edge starting from s_1. The sub-automaton $\mathscr{B}$ contains two locations, q_0 is a start and monitored location than when $\mathscr{B}$ visits q_0, it resets $x_{\mathscr{B}}$. The atomic proposition p decorating the edge starting from q_1. On the other hand, the atomic proposition $\neg p$ decorating the edge from q_1 to q_0.*

Theorem 30. *[Ras99, HRS98] The RECA are closed under all boolean operations.*

Theorem 31. *[Ras99, HRS98] The emptiness problem for RECA is decidable and PSPACE-Complete.*

Theorem 32. *[Ras99, HRS98] The problem of universality for RECA is decidable and PSPACE-Complete.*



xB< 3
S0
S1
Main automaton only reads clocks
p
Q0
Q1
¬p
Lower-level automaton B resets its clocks when visiting a monitored state or transition

Figure 5.15: Recursive Event Clock Automata

5.6 Distributed Timed Automata

Distributed Timed Automata (DTA) [Kri99, ABG+08] consist of several locals TA, called processes. Each process has its clocks. The clocks of a process evolve synchronously, but independently of the clocks of the other processes. The idea is that the clocks of the same process are all computed from the same hardware clock. A clock can be read by any process, but can only be reset by its owner process. The homonymous DTA of [DL07] work differently: they model processes whose execution is interleaved by a scheduler. Thus, only one process increases its (perfect) clocks at a time. They are a subclass of stopwatch automata. The product of DTA is studied in [ABG+08]. Instead, their product is first computed, giving rise to the class of Timed Automata with independent clocks (icTA).

Definition 55. *A DTA is a tuple $\mathscr{D} = (Proc, \mathscr{A}, \pi)$, such that :*

(i) Proc is a non-empty, finite set of process labels.

(ii) $\mathscr{A}$ is an indexed set of TA (see Definition 5.3.2), $\mathscr{A} = (\mathscr{A}_q)_{q \in Proc}$.

(iii) $\pi : \bigcup_{q \in Proc} X_q \to Proc$ maps each clock to its owner process.

where $\mathscr{A}_q$ can only reset its own clocks and $\mathscr{A}_q = (S_q, s^0_q, \Sigma_q, X_q, \to_{ta}, I_q, F_q)$ are TA.

Note that a process can read a clock of another process since the X_q need not be disjoint. In [ABG+08], DTA are not much studied. Instead, their product is first computed, giving rise to the class of TA with independent clocks (icTA). icTA assume a signature. A signature is a pair $(Proc, \mathbb{P})$, where *Proc* is a nonempty finite set of process labels, and $\mathbb{P}$ is a finite set of propositional symbols, from which we define $\Sigma = 2^{\mathbb{P}}$.

5.6.1 Timed Automata with Independent Clocks icTA

In [ABG+08], DTA are not much studied. Instead, their product is first computed, giving rise to the class of icTA. icTA were described in [ABG+08] and are neither determinizable nor complementable. Their emptiness problem can be solved using the region construction [ABG$^+$08], but their universal and inclusion problems are undecidable.

Definition 56 (**icTA** [ABG$^+$08])**.** *An icTA is a pair $\mathscr{A} = (\mathscr{B}, \pi)$ over Proc where $\mathscr{B}$ is a TA and $\pi : X \to Proc$ maps each clock to a process.*

Definition 57 (**Rates** [ABG$^+$08])**.** *A rate is a tuple $\tau = (\tau_q)_{q \in Proc}$ of local time functions. Each local time function τ_q maps the reference time to the time of process q, i.e, $\tau_q : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. The functions τ_q must be continuous, strictly increasing, divergent, and satisfy $\tau_q(0) = 0$.*

Note that the reference time is arbitrary, and thus not meaningful.

Definition 58. *Given a clock valuation $\nu : X \to \mathbb{R}_{\geq 0}$, a rate τ, and two reference times $t_1 > t_2$, the valuation $\nu + (t_1 - t_2)$ maps x to $\nu(x) + \tau_{\pi(x)}(t_1) - \tau_{\pi(x)}(t_2)$.*

The operational semantics of an icTA with pointwise semantics has been associated to a TLTS (i.e., a single-timed semantics [ABG$^+$08]).

Definition 59. *Given an icTA $\mathscr{A}$, a finite run of $\mathscr{A}$ for $\tau \in Rates$ is a path ρ of TLTS($\mathscr{A}$) starting from the initial state $q_0 = (s_0, \nu_0)$, with delay and discrete transitions alternating along the path: $\rho = (s_0, \nu_0) \xrightarrow{t_1, a_1} (s_1, \nu_1) \xrightarrow{t_2, a_2} (s_2, \nu_2) \ldots (s_{n-1}, \nu_{n-1}) \xrightarrow{t_n, a_n} (s_n, \nu_n)$ where $\forall\ 0 \leq i \leq n$, $s_i \in S$ and $\forall\ 1 \leq j \leq n$, $t_j \in \mathbb{R}_{\geq 0}$ and $w = a_j \in \Sigma$. A finite run of an icTA $\mathscr{A}$ over an untimed word is called an accepting run, iff $s_n \in F$ and $w \in \mathscr{L}$(TLTS($\mathscr{A}$)).*

In [ABG$^+$08] (p. 128) has been proved that $\mathscr{L}(\mathsf{TLTS}(\mathscr{A})) = \mathscr{L}_{\exists}(\mathscr{A})$, where $\mathscr{L}_{\exists}(\mathscr{A})$ is the existential language which is defined as $\mathscr{L}_{\exists}(\mathscr{A}) = \bigcup_{\tau \in Rates} \mathscr{L}(\mathscr{A}, \tau)$ and the universal language is defined as $\mathscr{L}_{\forall}(\mathscr{A}) = \bigcap_{\tau \in Rates} \mathscr{L}(\mathscr{A}, \tau)$. $\mathscr{L}(\mathscr{A}, \tau)$ is defined as the set of untimed words $\sigma \in \Sigma^*$ of accepting runs of $\mathscr{A}$ for τ.

Definition 60. *The language $\mathscr{L}(\mathscr{B}, \tau)$ is the set of TT of accepting runs of $\mathscr{B}$ for τ.*

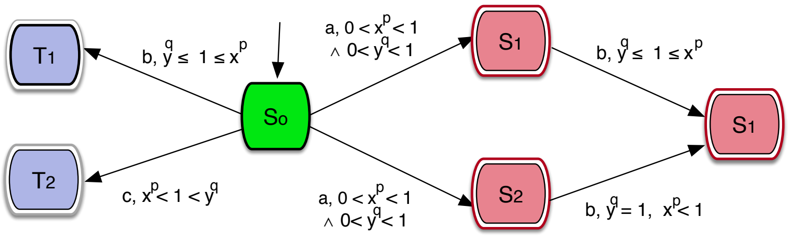


Figure 5.16: Example of an icTA $\mathscr{B}$ from [ABG$^+$08]

Example 17. *Let us consider the icTA $\mathscr{B}$ of the Figure 5.16. This icTA $\mathscr{B}$ contains the set of processes $\{p, q\}$, the automaton contains 6 locations, $\Sigma = \{a, b, c\}$, s_0 is the initial location, and also we assume $\pi(x) = p$ and $\pi(y) = q$. If both clocks are completely synchronized, they follow the same local clock rate (i.e, $\tau_p = \tau_q$), then the runs are* *those of a standard* TA. *If the clocks are synchronized, then the language is* $\mathscr{L}(\mathscr{B}, id) = \{a, b, ab\}$*, where* id *is to identify on* $\mathbb{R}_{\geq 0}$ *for all* $p \in Proc$*. If the clock* y *runs slower than clock* x *(i.e,* $\tau'_q \leq \tau'_p$*), the language is* $\mathscr{L}(\mathscr{B}, \tau') = \{a, ab, b\}$*. The existential language is* $\mathscr{L}_{\exists}(\mathscr{B}) = \{a, ab, b, c\}$ *and the universal language is* $\mathscr{L}_{\forall}(\mathscr{B}) = \{a, ab\}$*.*

5.7 Wrap up

This chapter presents the necessary background on distributed and real-time temporal formalisms that will be used in this thesis. We have reviewed the definition of the main formalisms proposed in the literature for modeling and specifying distributed and real-time systems. As we have already mentioned, distributed and real-time systems can be modeled using discrete and dense time semantics. In this thesis, we focus on the more interesting case where time is modeled by dense time semantics. In particular, we focus on the pointwise and continuous semantics. After defining the two time semantics, we reviewed the modal and real-time temporal logics. Then we reviewed the formalism of TA and its extensions (i.e., ECA, DTA, icTA).

Part II

Contributions and Results

“The eternal mystery of the world is its comprehensibility.”

— Albert Einstein

CHAPTER 6

DISTRIBUTED EVENT CLOCKS

In this chapter, we define the formalisms of Distributed Event Clock Automata (DECA) and (Recursive) Distributed Event Clock Temporal Logic (DECTL) in the context of TIS for DRTS. DECA is a variant of RECA inspired by Distributed Timed Automata (DTA) and Timed Automata with Independent Clocks (icTA) and proposed by [ABG$^+$08, DL07, Kri99] to model DRTS. In DECA and DECTL, the clocks can advance independently if they are in different processes. DECA are closed under all boolean operations. Also, the emptiness and universality problems are decidable, allowing stepwise refinement. We have chosen to develop the details in continuous semantics because we will relate the result obtained in this chapter to the logics EventClockTL [HRS98]. Satisfiability and validity problems for DECTL are decidable. This logic can be model-checked by translating a DECTL formula into a DECA automaton.

This chapter is structured as follows. In section 6.1, we address the limitation presented in the abstract of this thesis and introduce Distributed Event Clocks (DEC). We extend Recursive Event Clock Automata (RECA) with such distributed (a.k.a. independent) event clocks, resulting in Distributed Recursive Event Clock Automata (DECA). We will show that DECA are determinizable, i.e. closed under complementation, and thus that their language inclusion problem is decidable (more precisely, PSPACE-complete). We also show the decidability and regularity of their universal languages. In section 6.2 we study the corresponding timed languages of DTA and icTA. In section 6.3 we extend EventClockTL with distributed clocks. This gives us the (Recursive) Distributed Event Clock Temporal Logic (DECTL), which we show to be PSPACE-complete. In section 6.4 we present some scenarios of distributed real-time systems for DECA and DECTL. Finally, in section 6.5 we show the strengths and weaknesses of DECA and DECTL.

6.1 Distributed Event Clock Automata

In this section, we focus on restoring full decidability of icTA [ABG$^+$08], we use event clocks [AH91]. For expressiveness, we use RECA [HRS98] with independent clocks [ABG$^+$08]. The event clock $x^q_{\mathscr{A}}$ (or $y^q_{\mathscr{A}}$) denote records the time since the last (resp. next) time that the automaton $\mathscr{A}$ could visit a monitored state or transition, measured in the local time of process q.

6.1.1 Distributed Event Clocks

The definition of Distributed Event Clocks (DEC) is the same as for RECA 5.5, but each clock is associated with a process (i.e., $Proc = \{p, q, \ldots\}$ is a set of processes). Let $\mathscr{B}$ be a lower-level automaton. Let $Proc$ be a set of processes. Let $X = \{R, P\}$ be a finite set of DEC associated with the lower-level automaton, $\mathscr{B}$ where $R = \{x^q_{\mathscr{B}} \mid \mathscr{B}$ is a lower-level automaton and $q \in Proc\}$, is the set of Distributed Event Recording Clocks (DERC) and $P = \{y^q_{\mathscr{B}} \mid \mathscr{B}$ is a lower-level automaton and $q \in Proc\}$, is the set of Distributed Event Predicting Clocks (DEPC). Here, we will use non-standard clock values, as in RECA 5.5. In the following, we note $z^q_{\mathscr{B}}$ each recording and predicting clock of X associated with the lower-level automaton $\mathscr{B}$, measured in the local time of the process q.

6.1.2 Distributed Event Clock Invariants

Let $\mathscr{B}$ be a lower-level automaton. Let *Proc* be a set of processes. Let X be a finite set of DEC ranging over $\mathbb{R}_{\geq 0}$. Let $\Delta(X)$ be a set of distributed event clock invariants. Distributed event clock invariants are event clock constraints of the following form:

$$\varphi := true \mid z^q_{\mathscr{B}} < c \mid z^q_{\mathscr{B}} \leq c \mid \varphi_1 \wedge \varphi_2$$

where $z^q_{\mathscr{B}} \in X$, $c \in \mathbb{N}$ and $q \in Proc$.

6.1.3 Distributed Event Clock Constraints

Let $\mathscr{B}$ be a lower-level automaton. Let *Proc* be a set of processes. Let X be a finite set of DEC associated to Σ. The set $\Phi(X)$ of distributed event clock constraints over the set of clocks X is given by the following grammar:

$$\phi := true \mid z^q_{\mathscr{B}} \sim c \mid \phi_1 \wedge \phi_2$$

where $z^q_{\mathscr{B}} \in X$, $c \in \mathbb{N}$, $\sim\, \in \{<, >, \leq, \geq, =\}$ and $q \in Proc$.

6.1.4 Distributed Event Clock Valuations

Let $\mathscr{B}$ be a lower-level automaton. Let X be a finite set of event clocks. A clock valuation function, $\nu : X \to (\mathbb{R}_{\geq 0} \cup \{\perp\})$ assigning to each event clock $x \in X$ a value in $(\mathbb{R}_{\geq 0} \cup \{\perp\})$. Given an TIS ρ, a rate τ and time $t \in \mathbb{R}_{\geq 0}$, a clock valuation $\nu(\rho, t, \tau) \in (\mathbb{R}^X_{\geq 0} \cup \{\perp\})$ over X, specifies the values of distributed event clocks in X at time t, along ρ on the rate τ.

Given an TIS ρ, the valuation of the DERC variable $x^q_{\mathscr{B}}$ along a ρ at time t and on the rate τ is:

$$\nu(\rho, t, \tau, x^q_{\mathscr{B}}) = \begin{cases} \tau_q(t) - \tau_q(r) & \text{if } r = \max\{t' < t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B}, \tau)\} \text{ exists} \\ (\tau_q(t) - \tau_q(r))^+ & \text{else, if } r = \sup\{t' < t \mid (t', \rho) \in \mathscr{L}^+(\mathscr{B}, \tau)\} \text{ exists} \\ \perp & \text{else} \end{cases}$$

The definition for DEPC is symmetric.

$$\nu(\rho, t, \tau, y^q_{\mathscr{B}}) = \begin{cases} \tau_q(l) - \tau_q(t) & \text{if } l = \min\{t' > t | (t', \rho) \in \mathscr{L}^+(\mathscr{B}, \tau)\} \text{ exists} \\ (\tau_q(l) - \tau_q(t))^+ & \text{else, if } l = \inf\{t' > t \,|(t', \rho) \in \mathscr{L}^+(\mathscr{B}, \tau)\} \text{ exists} \\ \perp & \text{else} \end{cases}$$

Definition 61. *A Distributed Recursive Event Clock Automaton (DECA) is a pair $(\mathscr{A}, \pi)$ where $\mathscr{A}$ is a RECA (see Definition 5.5) and $\pi : X \to Proc$ maps each clock to a process.*

For better readability, we write the owner process in the clock name: $\pi(x^q_{\mathscr{A}}) = q$.

Definition 62. *A run θ of a DECA $\mathscr{A}$ for a rate τ is a pair of sequences (s, I) where s gives an alternation of transitions and locations $\zeta_0, s_1, \zeta_2, s_2, \ldots$, and I is an interval sequence, such that:*

(i) The run starts from the initial state: $\zeta_0 \in S_0 \times S$.

(ii) For all i > 1, the run follows a discrete transition: $\zeta_i = (s_i, s_{i+1}) \in \to_{deca}$

(iii) The clock constraints (invariant or guard) are satisfied by the valuation of the clocks defined above: $\forall\, t \in \mathbb{R}_{\geq 0}, \nu(\rho, t, \tau) \models Inv(\theta(t))$.

(iv) It satisfies the acceptance condition, e.g. it visits infinitely often an accepting location.

The TIS of a run $\theta = (s, I)$ of a DECA $\mathscr{A}$, noted $\gamma(\theta)$ is the pair $(\gamma(s), I)$. We say that $\mathscr{A}$ accepts an TIS ρ at t with τ, if there is a run θ for an equivalent of ρ that visits a monitored location (or transition) at t. This is noted by $(t, \rho) \in \mathscr{L}^+(\mathscr{A}, \tau)$, its timed language. This time t will be used to reset the associated clocks $x^q_{\mathscr{B}}$ above.

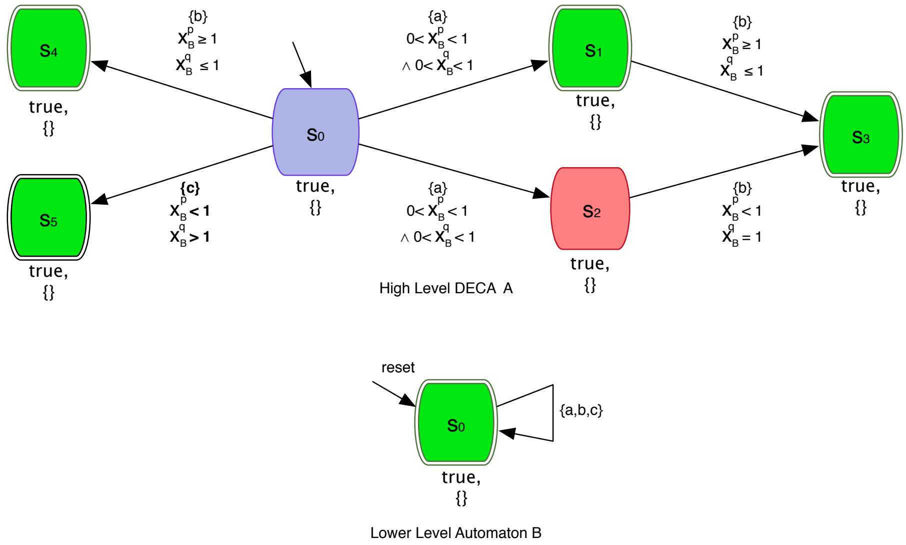


Figure 6.1: Example of DECA from [ABG⁺08]

Example 18. *The example of Figure 6.1 from [ABG⁺ 08] is in fact both a DECA and an icTA $\mathscr{A}$ over Proc = $\{p, q\}$, and the set of propositions $\mathbb{P} = \{a, b, c\}$. States have an empty labelling. Both clocks are reset by the initial monitored transition of $\mathscr{B}$. After this, they may diverge. The existential timed language, here, is read from the automaton:*

$$\begin{aligned}
\mathscr{L}_{\exists}(\mathscr{A}, p) &= \mathit{ITL}^{1}(\{(a, t_1^p) \mid 0 < t_1^p < 1\} \cup \{(b, t_1^p) \mid t_1^p \geq 1\} \cup \{(c, t_1^p) \mid 0 < t_1^p < 1\} \\
&\quad \cup \{(a, t_1^p), (b, t_2^p) \mid 0 < t_1^p < 1 \wedge t_1^p < t_2^p\}) \\
\mathscr{L}_{\exists}(\mathscr{A}, q) &= \mathit{ITL}(\{(a, t_1^q) \mid 0 < t_1^q < 1\} \cup \{(b, t_1^q) \mid 0 < t_1^q \leq 1\} \cup \{(c, t_1^q) \mid t_1^q > 1\} \\
&\quad \cup \{(a, t_1^q), (b, t_2^q) \mid 0 < t_1^q < 1 \wedge t_1^q < t_2^q \leq 1\})
\end{aligned}$$

Here, all universal timed languages are empty: $\mathscr{L}_{\forall}(\mathscr{A}, p) = \emptyset = \mathscr{L}_{\forall}(\mathscr{A}, q)$. For instance, we cannot have $(a, t_a) \in \mathscr{L}_{\forall}(\mathscr{A}, p)$, because there are some τ where the time of q increases steeply, and gets over 1 before the time of p could reach t_a. However, the universal untimed language $\mathscr{L}_{\forall}(\mathscr{A})$ is $\{a, ab\}$.

6.1.5 Multi-Timed Languages of DECA

DECA inherit the main property of RECA: They are determinizable. Determinization preserves the τ-wise, existential and universal languages, and there are closures over boolean operations. The theorems below are valid for the finite version, but also for the infinite version, e.g. for Büchi automata, which are determinized to a parity automaton [Pit06]. For better readability, we use the notations ν and $\nu(\rho, t, \tau)$ to denote distributed event clock valuations.

[1] ITL will add the missing intervals between time points.

Definition 63. *A DECA $\mathscr{A}$ is deterministic iff all the following conditions holds:*

(i) $\mathscr{A}$ has exactly one initial location $\{s_0\} = S_0$ and,

(ii) It has no ϵ-transitions: There are no two successive locations $s_1 \rightarrow s_2$, with the same labeling: $\gamma(s_1) = \gamma(s_1, s_2) = \gamma(s_2)$ and,

(iii) Any two distinct successor locations $s_2 \neq s_3, s_1 \rightarrow s_2, s_1 \rightarrow s_3$ with same labeling: $\gamma(s_2) = \gamma(s_3)$ and $\gamma(s_1, s_2) = \gamma(s_1, s_3)$, have mutually exclusive clock constraints: $v \not\models Inv(s_1, s_2) \wedge Inv(s_1, s_3)$.

Definition 64. *A DECA $\mathscr{A}$ is complete iff: for any symbol $\sigma \in \Sigma$, any clock valuation v, and for any location $s \in S$ there is a successor location $s_1 \in S$ with $\gamma(s_1) = \sigma$ and $v \models Inv(s_1)$.*

Therefore, if $\mathscr{A}$ is a complete DECA, then for every TIS ρ there is at least one accepting run θ of $\mathscr{A}$. The determinism ensures that at any time t during a run, the choice of the next state is uniquely determined by the current location of the automaton and (ρ, τ). Note that we have imposed a disjoint label from each state to the next to ensure that the time at which to leave a state is unique and given by ρ. Therefore, there is at most a single run for each ρ. Below is a proof of the construction of a deterministic Rabin (DR) $\mathscr{A}$ (DR($\mathscr{A}$)) from a DECA $\mathscr{A}$. This proof is an adaptation of the construction of RECA [HRS98].

Theorem 33. *For any DECA $\mathscr{A}$, we construct a DR $\mathscr{B}$ ($\mathscr{B}$ = DR($\mathscr{A}$)) that accepts the same language.*

Proof. Given $\mathscr{A}$, we construct the DR $\mathscr{B}$ = DR($\mathscr{A}$) as follows:

(i) The set of locations of $\mathscr{B}$ is the set of non-empty subsets of locations of $\mathscr{A}$ with the same labelling, that is $\{s_1, s_2, \cdots, s_n\} \in S^{\mathscr{B}}$ iff :
  (a) $n \geq 1$,
  (b) for all i, $1 \leq i \leq n$: $s_i \in S^{\mathscr{A}}$,
  (c) for all i, j such that $1 \leq i < j \leq n$, we have that $\gamma^{\mathscr{A}}(s_i) = \gamma^{\mathscr{A}}(s_j)$.

(ii) The set of starting location of $\mathscr{B}$ is the subset of locations that contains only initial location of $\mathscr{A}$, that is, $q = s_0^{\mathscr{B}}$ where $q \in S_0^{\mathscr{B}}$, iff:
  (a) for all $s \in q$, where $s \in S_0^{\mathscr{A}}$,
  (b) it does not exist a location q' with
    i. $\gamma^{\mathscr{B}}(q') = \gamma^{\mathscr{B}}(q)$,
    ii. for all $s \in q'$, then $s \in S_0^{\mathscr{A}}$,
    iii. $q \subset q'$.

(iii) The set of propositions used in $\mathscr{B}$ is the same as the set of propositions used in $\mathscr{A}$: $\Sigma^{\mathscr{B}} = \Sigma^{\mathscr{A}}$,

(iv) The set of clocks used in $\mathscr{B}$ is the same as the set of clocks used in $\mathscr{A}$, $X^{\mathscr{B}} = X^{\mathscr{A}}$.

(v) The labeling function $\gamma^{\mathscr{B}}(q) = \gamma^{\mathscr{A}}(s)$ with $s \in q$, for all $q \in S^{\mathscr{B}}$. The locations $q \in S^{\mathscr{B}}$ are labelled with the same labels as in $\mathscr{A}$.

(vi) The labeling function for a transition relation $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca}$ and $(s_1, s_2) \in \rightarrow^{\mathscr{A}}_{deca}$, $\gamma^{\mathscr{B}}((q_1, q_2)) = \gamma^{\mathscr{A}}((s_1, s_2))$ with $s_1 \in q_1$ and $s_2 \in q_2$, for all $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$. The transitions $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$ are labelled with the same labels as in $\mathscr{A}$.

(vii) The invariant clock constraint $inv^{\mathscr{B}}(q) = \bigwedge inv^{\mathscr{A}}(s)$ with $s \in q$, for all $q \in S^{\mathscr{B}}$. The locations $q \in S^{\mathscr{B}}$ are assigned with the conjunction of all the clock constraints of $s \in q$.

(viii) The invariant clock constraint for a transition relation $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca}$ and $(s_1, s_2) \in \rightarrow^{\mathscr{A}}_{deca}$, $inv^{\mathscr{B}}((q_1, q_2)) = \bigwedge inv^{\mathscr{A}}((s_1, s_2))$ with $s_1 \in q_1$ and $s_2 \in q_2$, for all $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$. The transitions $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$ are assigned with the conjunction of all the clock constraints of $s_1 \in q_1$ and $s_2 \in q_2$.

(ix) A transition relation, we define $(q_1, q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$ iff:
   (a) for all $s_2 \in q_2$, there exists $s_1 \in q_1$ such that, $(s_1, s_2) \in \rightarrow^{\mathscr{A}}_{deca}$; the locations in q_2 are $\rightarrow^{\mathscr{A}}$-successors of locations in q_1,
   (b) for all $s_2 \in S^{\mathscr{A}}$ such that $\gamma^{\mathscr{A}}(s_2) = \gamma^{\mathscr{B}}(q_2)$ and there exists $s_1 \in q_1$ with $(s_1, s_2) \in \rightarrow^{\mathscr{A}}_{deca}$, we have $s_2 \in q_2$, i.e. q_2 is the maximal set of locations that share the label of q_2 and are $\rightarrow^{\mathscr{A}}_{deca}$-successors of a location of q_1.

(x) A monitored location $q \in S^{\mathscr{B}}$ belongs to the set $M^{\mathscr{B}}$ of monitored transitions iff there exists a location of $\mathscr{A}$ in q that is monitored, i.e. $q \in M^{\mathscr{B}}$ iff there exists $s \in q$ such that $s \in M^{\mathscr{A}}$.

(xi) A monitored transition $(q_1,\ q_2) \in \rightarrow^{\mathscr{B}}_{deca} \subseteq S^{\mathscr{B}} \times S^{\mathscr{B}}$ belongs to the set $M^{\mathscr{B}}$ of monitored locations iff there exists a transition of $\mathscr{A}$ in $(q_1,\ q_2)$ that is monitored, i.e. $(q_1,\ q_2) \in M^{\mathscr{B}}$ iff there exists $s_1 \in q_1$ and $s_2 \in q_2$ such that $(s_1,\ s_2) \in M^{\mathscr{A}}$.

(xii) The accepting condition $F^{\mathscr{B}} = \{(L_1, U_1, \ldots, L_{2n}, U_{2n})\}$ consists of $2n$ acceptance pairs, defined as: (1) L_i are the set of states visited finitely often. (2) U_i are the set of are visited infinitely often.

(xiii) The function that maps each clock to a process in $\mathscr{B}$ is $\pi^{\mathscr{B}} = \pi^{\mathscr{A}}$.

□

Theorem 34. *DECA is closed under union operation.*

Proof. Let $\mathscr{A} = (S^{\mathscr{A}}, S^{\mathscr{A}}_0, \Sigma^{\mathscr{A}}, X^{\mathscr{A}}, \gamma^{\mathscr{A}}, inv^{\mathscr{A}}, \rightarrow^{\mathscr{A}}_{deca}, M^{\mathscr{A}}, F^{\mathscr{A}}, \pi^{\mathscr{A}})$ and $\mathscr{B} = (S^{\mathscr{B}}, S^{\mathscr{B}}_0, \Sigma^{\mathscr{B}}, X^{\mathscr{B}}, \gamma^{\mathscr{B}}, inv^{\mathscr{B}}, \rightarrow^{\mathscr{B}}_{deca}, M^{\mathscr{B}}, F^{\mathscr{B}}, \pi^{\mathscr{B}})$ be two DECA. Without loss of generality, we assume that the sets of clocks $X^{\mathscr{A}}$ and $X^{\mathscr{B}}$ (and respectively the sets of locations $S^{\mathscr{A}}$ and $S^{\mathscr{B}}$) are all pairwise disjoint. Let $\mathscr{C} = (S^{\mathscr{C}}, S^{\mathscr{C}}_0, \Sigma^{\mathscr{C}}, X^{\mathscr{C}}, \gamma^{\mathscr{C}}, \delta^{\mathscr{C}}, \rightarrow^{\mathscr{C}}_{deca}, M^{\mathscr{C}}, F^{\mathscr{C}}, \pi^{\mathscr{C}})$ be the DECA defined as follows:

(i) The locations of $\mathscr{C}$ are tuples (s, μ) such that either:
   (a) $s \in S^{\mathscr{A}}$, $\mu \in (\Sigma^{\mathscr{C}} \cup inv^{\mathscr{C}})$ and for all $\varsigma \in (\Sigma^{\mathscr{A}} \cup inv^{\mathscr{A}}) : \varsigma \subseteq \mu$ iff $\varsigma \in \delta^{\mathscr{A}}(s)$, which will ensure the coherence of the labelling of (s, μ) with the labelling of s in $\mathscr{A}$, $S^{\mathscr{C}} = S^{\mathscr{A}} \cup S^{\mathscr{B}}$,
   (b) or $s \in S^{\mathscr{B}}$, $\mu \in (\Sigma^{\mathscr{C}} \cup inv^{\mathscr{C}})$ and for all $\varsigma \in (\Sigma^{\mathscr{B}} \cup inv^{\mathscr{B}})$: $\varsigma \subseteq \mu$ iff $\varsigma \in inv^{\mathscr{A}}(s)$, which will ensure the coherence of the labelling of (s, μ) with the labelling s in $\mathscr{B}$,

(ii) The starting location of $\mathscr{C}$ is the following $S^{\mathscr{C}}_0 = \{(s, \mu) \in S^{\mathscr{C}} \mid s = S^{\mathscr{A}}_0 \ or \ s = S^{\mathscr{B}}_0)\}$,

(iii) The alphabet in $\Sigma^{\mathscr{C}}$ is as in $\mathscr{A}$, that is $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} = \Sigma^{\mathscr{B}}$, $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} \cup \Sigma^{\mathscr{B}}$,

(iv) The clocks of $\mathscr{C}$ are the disjoint union of $\mathscr{A}$ and $\mathscr{B}$, that is $X^{\mathscr{C}} = X^{\mathscr{A}} \cup X^{\mathscr{B}}$,

(v) The subset of monitored locations of $\mathscr{C}$ is the following set: $M^{\mathscr{C}} = \{(s, \mu) \in S^{\mathscr{C}} \mid s \in M^{\mathscr{A}} \ or \ s \in M^{\mathscr{B}}\}$,

(vi) The label of the location (s, μ) is simply the set of symbols $\xi : \gamma^{\mathscr{C}}((s, \mu)) = \xi$, where $\xi \in \Sigma^{\mathscr{C}}$, for every $(s, \mu) \in S^{\mathscr{C}}$,

(vii) The clock constraints of the locations in $\mathscr{C}$ is the union of the clock constraints of the location s^a in $\mathscr{A}$ and the location s^b in $\mathscr{B}$, that is $inv^{\mathscr{C}}((s^a, s^b)) = inv^{\mathscr{A}}(s^a) \cup inv^{\mathscr{B}}(s^b)$,

(viii) The transition relation of $\mathscr{C}$ is the following subset of $S^{\mathscr{C}} \times S^{\mathscr{C}} : \rightarrow^{\mathscr{C}}_{deca} = \{[(s_1, \mu_1), (s_2, \mu_2)] \mid (s_1, s_2) \in \rightarrow^{\mathscr{A}}_{deca} \ or \ (s_1, s_2) \in \rightarrow^{\mathscr{B}}_{deca}\}, \rightarrow^{\mathscr{C}}_{deca} = \rightarrow^{\mathscr{A}}_{deca} \cup \rightarrow^{\mathscr{B}}_{deca},$

(ix) The accepting conditions for $\mathscr{C}$ is the union of the accepting condition for $\mathscr{A}$ and $\mathscr{B}$, that is $F^{\mathscr{C}} = \{(s, \mu) \mid s \in F^{\mathscr{A}} \ or \ s \in F^{\mathscr{B}}\}$. $F^{\mathscr{C}} = F^{\mathscr{A}} \cup F^{\mathscr{B}}$,

(x) The function that maps each clock to a process in $\mathscr{C}$ is the union of $\mathscr{A}$ and $\mathscr{B}$, that is $\pi^{\mathscr{C}} = \pi^{\mathscr{A}} \cup \pi^{\mathscr{B}}$.

□

Theorem 35. *DECA is closed under intersection operation.*

Proof. Let $\mathscr{A} = (S^{\mathscr{A}}, S_0^{\mathscr{A}}, \Sigma^{\mathscr{A}}, X^{\mathscr{A}}, \gamma^{\mathscr{A}}, inv^{\mathscr{A}}, \rightarrow^{\mathscr{A}}_{deca}, M^{\mathscr{A}}, F^{\mathscr{A}}, \pi^{\mathscr{A}})$ and $\mathscr{B} = (S^{\mathscr{B}}, S_0^{\mathscr{B}}, \Sigma^{\mathscr{B}}, X^{\mathscr{B}}, \gamma^{\mathscr{B}}, inv^{\mathscr{B}}, \rightarrow^{\mathscr{B}}_{deca}, M^{\mathscr{B}}, F^{\mathscr{B}}, \pi^{\mathscr{B}})$ be two DECA. Without loss of generality, we assume that the sets of clocks $X^{\mathscr{A}}$ and $X^{\mathscr{B}}$ (and respectively the sets of locations $S^{\mathscr{A}}$ and $S^{\mathscr{B}}$) are all pairwise disjoint. Let $\mathscr{C} = (S^{\mathscr{C}}, S_0^{\mathscr{C}}, \Sigma^{\mathscr{C}}, X^{\mathscr{C}}, \gamma^{\mathscr{C}}, inv^{\mathscr{C}}, \rightarrow^{\mathscr{C}}_{deca}, M^{\mathscr{C}}, F^{\mathscr{C}}, \pi^{\mathscr{C}})$ be the DECA defined as follows:

(i) The set of locations of $\mathscr{C}$ are the tuples (s^a, s^b) such that $s^a \in S^{\mathscr{A}}$, $s^b \in S^{\mathscr{B}}$ and for all $\varsigma \in (\Sigma^{\mathscr{A}} \cup \delta^{\mathscr{A}}) \cap (\Sigma^{\mathscr{B}} \cup inv^{\mathscr{B}})$, $\varsigma \in inv^{\mathscr{A}}(s^a)$ iff $\varsigma \in inv^{\mathscr{B}}(s^b)$, $S^{\mathscr{C}} = S^{\mathscr{A}} \times S^{\mathscr{B}}$,

(ii) The starting location of $\mathscr{C}$ is the following $S_0^{\mathscr{C}} = \{(s^a, s^b) \in S^{\mathscr{C}} \mid s^a = S_0^{\mathscr{A}} \ and \ s^b = S_0^{\mathscr{B}}\}$,

(iii) The alphabet in $\Sigma^{\mathscr{C}}$ is as in $\mathscr{A}$, that is $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} = \Sigma^{\mathscr{B}}$, $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} \cap \Sigma^{\mathscr{B}}$,

(iv) The clocks of $\mathscr{C}$ are the union of $\mathscr{A}$ and $\mathscr{B}$, that is $X^{\mathscr{C}} = X^{\mathscr{A}} \cup X^{\mathscr{B}}$,

(v) The subset of monitored locations of $\mathscr{C}$ is the following set: $M^{\mathscr{C}} = \{(s^a, s^b) \in S^{\mathscr{C}} \mid s^a \in M^{\mathscr{A}} \ and \ s^b \in M^{\mathscr{B}}\}$,

(vi) The label locations (s^a, s^b) of $\mathscr{C}$ is the intersection of the label of s^a in $\mathscr{A}$ and the label of s^b in $\mathscr{B}$, that is $\gamma^{\mathscr{C}}((s^a, s^b)) = \gamma^{\mathscr{A}}(s^a) \wedge \gamma^{\mathscr{B}}(s^b)$, for every $(s^a, s^b) \in S^{\mathscr{C}}$, $\gamma^{\mathscr{C}} = \gamma^{\mathscr{A}} \cup \gamma^{\mathscr{B}}$,

(vii) The transition relation of $\mathscr{C}$ is the following set $S^{\mathscr{C}} \times S^{\mathscr{C}} : \rightarrow^{\mathscr{C}}_{deca} = \{[(s_1^a, s_1^b), (s_2^a, s_2^b)] \mid (s_1^a, s_2^a) \in \rightarrow^{\mathscr{A}}_{deca} \vee (s_1^a = s_2^a) \ and \ (s_1^b, s_2^b) \in \rightarrow^{\mathscr{B}}_{deca} \vee (s_1^b = s_2^b)\}$, $((s_1^{\mathscr{A}}, s_2^{\mathscr{A}}), (s_1^{\mathscr{B}}, s_2^{\mathscr{B}})) \in \rightarrow^{\mathscr{C}}_{deca}$ iff there exist transitions $(s_1^{\mathscr{A}}, s_2^{\mathscr{A}}) \in \rightarrow^{\mathscr{A}}_{deca}$ and $(s_1^{\mathscr{A}}, s_2^{\mathscr{A}}) \in \rightarrow^{\mathscr{B}}_{deca}$,

(viii) The clock constraints of the locations of $\mathscr{C}$ is the intersection of the clock constraints of the location s^a in $\mathscr{A}$ and the location s^b in $\mathscr{B}$, that is $inv^{\mathscr{C}}((s^a, s^b)) = inv^{\mathscr{A}}(s^a) \wedge inv^{\mathscr{B}}(s^b)$, $inv^{\mathscr{C}} = inv^{\mathscr{A}} \wedge inv^{\mathscr{B}}$,

(ix) $((s_{0_1}^{\mathscr{A}}, s_{0_2}^{\mathscr{A}}), (s_{0_1}^{\mathscr{B}}, s_{0_2}^{\mathscr{B}})) \in \mathscr{C}$ iff there exist transitions $(s_{0_1}^{\mathscr{A}}, s_{0_2}^{\mathscr{A}}) \in \rightarrow_0^{\mathscr{A}}$ and $(s_{0_1}^{\mathscr{A}}, s_{0_2}^{\mathscr{A}}) \in \rightarrow_0^{\mathscr{B}}$,

(x) The accepting conditions for $\mathscr{C}$ is defined using a generalized Büchi condition : $F^{\mathscr{C}} = \{G_{\mathscr{A}}, G_{\mathscr{B}}\}$ with $G_{\mathscr{A}} = \{(s^a, s^b) \mid s^a \in F^{\mathscr{A}}\}$ and $G_{\mathscr{B}} = \{(s^a, s^b) \mid s^b \in F^{\mathscr{B}}\}$ and the reduction from a generalized Büchi automata to Büchi automata is : $F^{\mathscr{C}} = F^{\mathscr{A}} \times \{1\}$.

(xi) The function that maps each clock to a process in $\mathscr{C}$ are the union of $\mathscr{A}$ and $\mathscr{B}$, that is $\pi^{\mathscr{C}} = \pi^{\mathscr{A}} \cup \pi^{\mathscr{B}}$.

□

Theorem 36. *DECA is closed under complementation operation.*

Proof. Given a DECA $\mathscr{A} = (S^{\mathscr{A}}, S_0^{\mathscr{A}}, \Sigma^{\mathscr{A}}, X^{\mathscr{A}}, \gamma^{\mathscr{A}}, inv^{\mathscr{A}}, \rightarrow^{\mathscr{A}}, M^{\mathscr{A}}, F^{\mathscr{A}}, \pi^{\mathscr{A}})$, such that, DECA are closed under complementation since they can be determinizable. Given DECA $\mathscr{A}$, we construct a Muller complement automaton of $\mathscr{A}$, such that, $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A})^c$. In the 63 and 33, we show that for an automaton $\mathscr{A}$, we can have a deterministic Rabin automaton Det($\mathscr{A}$), using Safra's Construction. Therefore, we can construct the Muller complement automaton of $\mathscr{A}$ as following : (*i*) We construct a determinist Rabin automaton Det($\mathscr{A}$) as in 63 and 33. (*ii*) A deterministic Rabin automaton Det($\mathscr{A}$) has a unique run corresponding to a given TIS ρ. (*iii*) If we have a deterministic Rabin automaton Det($\mathscr{A}$), we can easily translate into an equivalent deterministic Muller automaton Det($\mathscr{A}$). (*iv*) Hence, we have an equivalent deterministic Muller automaton Det($\mathscr{A}$) for which, we can complement replacing F by $S \backslash F$ as the final locations for the acceptance of $\mathscr{A}$ would result in the automaton accepting the complement of $\mathscr{L}(\mathscr{A})^c$. □

proposition 1. *For every DECA $\mathscr{A}$ and Det($\mathscr{A}$) with Rabin accepting condition, accepts the same language: $\mathscr{L}^+(\mathscr{A}, \tau) = \mathscr{L}^+(\text{Det}(\mathscr{A}), \tau)$ for all $\mathscr{A}$ and deterministic Det($\mathscr{A}$).*

Proof. The proof consists of two steps, first showing $\mathscr{L}^+(\mathscr{A}, \tau) \subseteq \mathscr{L}^+(\text{Det}(\mathscr{A}), \tau)$ and then showing $\mathscr{L}^+(\text{Det}(\mathscr{A}), \tau) \subseteq \mathscr{L}^+(\mathscr{A}, \tau)$.

(i) $\mathscr{L}^+(\mathscr{A}, \tau) \subseteq \mathscr{L}^+(\text{Det}(\mathscr{A}), \tau)$: Let ρ be an TIS. We show that there exists an accepting run θ for ρ at time t over $\mathscr{A}$ for a rate τ iff there exists an accepting run θ' for the same τ and ρ over deterministic Rabin automaton Det($\mathscr{A}$). We are using induction over the length of the runs. For θ' to be accepting, one of the acceptance pairs (L_j, U_j) has to be satisfied, i.e. from some point on the states from L_j are visited infinitely often and no states from U_j are visited, for $1 \leq i \leq 2n$. This means we have to find a state j in the automaton Det($\mathscr{A}$) that is visited infinitely often (so the states are included in L_j) and is never removed (so the states will not be in U_j). Let $\theta = (s_0, I_0), (s_1, I_1), (s_2, I_2), \ldots$ be the run for ρ at time t and on the *rate* τ over $\mathscr{A}$, where s are an alternation of transitions and locations $\delta_1, s_1, \delta_2, s_2, \cdots$, (where $\delta_i = (s_{i-1}, s_i) \in \rightarrow_{deca}$), and I_i is an alternating sequence of interval. By induction hypothesis this is possible only if there exists a run $\theta' = (s_0^{'}, I_0), (s_1^{'}, I_1), (s_2^{'}, I_2), \ldots$ for ρ at time t and on the *rate* τ over Det($\mathscr{A}$), where we can safely assume due to the construction that, for $i \geq 1$, $s_i \in s_i^{'}$. Firstly, we say that $\mathscr{A}$ accepts an TIS ρ at t with *rate* τ, if there is a run θ for ρ that visits infinitely often a monitored location at t. This time t will use to reset the clock. Secondly, the clock valuation depends on the TIS ρ, on the reference time of evaluation t, and on the *rate* τ. It is easy to see that for $i \geq 1$ the clock valuation assigns a (non-standard) positive real, or undefined, to each clock variable $\nu(\rho, t, \tau)_{\mathscr{A}} = \nu^{'}(\rho, t, \tau)_{\text{Det}(\mathscr{A})}$. Thirdly, since

$s_{i-1} \in s^{'}_{i-1}$ and there exists a transition δ_i from (s_{i-1}, s_i) in $\mathscr{A}$, there exists $s^{'}_i$ such that $s_i \in s^{'}_i$ and there exists a transition $\delta^{'}_i$ from $(s^{'}_{i-1}, s^{'}_i)$ where the clock constraints are satisfied by the valuation $v^{'}(\rho, t, \tau)_{\mathsf{Det}(\mathscr{A})}$. Hence, we can add $s^{'}_{i-1}$, $\delta^{'}_i$, $s^{'}_i$ to the run over $\mathsf{Det}(\mathscr{A})$. It follows immediately that $\mathscr{A}$ accepts the set of TIS ρ at time t and on the *rate* τ, then the language $\mathscr{L}^+(\mathscr{A}, \tau)$ is defined as the set of TIS ρ of accepting runs of $\mathscr{A}$ with regard to τ, and the language $\mathscr{L}^+(\mathsf{Det}(\mathscr{A}), \tau)$ is defined as the set of TIS ρ of accepting runs of $\mathsf{Det}(\mathscr{A})$ with regard to τ.

(ii) $\mathscr{L}^+(\mathsf{Det}(\mathscr{A}), \tau) \subseteq \mathscr{L}^+(\mathscr{A}, \tau)$: The other direction of the implication is trivial: If a run is accepted by a deterministic automaton, then it is accepted by a nondeterministic one.

□

proposition 2. *For every DECA $\mathscr{A}$, Det($\mathscr{A}$) accepts the same existential language $\mathscr{L}_\exists(Det(\mathscr{A}), P) = \mathscr{L}_\exists(\mathscr{A}, P)$ for any $P \subseteq Proc$ then for all $\mathscr{A}$ and deterministic Det($\mathscr{A}$).*

Proof. Let ρ be an TIS. We show that there exists an accepting run θ for ρ at time t over $\mathscr{A}$ for a rate τ iff there exists an accepting run $\theta^{'}$ for the same τ and ρ over $\mathsf{Det}(\mathscr{A})$. We are using induction over the length of the runs. Let $\theta = (s_0, I_0), (s_1, I_1), (s_2, I_2), \ldots (s_n, I_n)$ be the run for ρ at time t and on the *rate* τ over $\mathscr{A}$, where s are an alternation of transitions and locations $\delta_1, s_1, \delta_2, s_2, \cdots$, (where $\delta_i = (s_{i-1}, s_i) \in \rightarrow_{deca}$), and I_i is an alternating sequence of interval. By induction hypothesis this is possible only if there exists a run $\theta' = (s^{'}_0, I_0), (s^{'}_1, I_1), (s^{'}_2, I_2), \ldots (s^{'}_n, I_n)$ for ρ at time t and on the *rate* τ over $\mathsf{Det}(\mathscr{A})$, where we can safely assume due to the construction that, for each $1 \le i \le n$, $s_i \in s^{'}_i$. Firstly, we say that $\mathscr{A}$ accepts an TIS ρ at t with *rate* τ, if there is a run θ for ρ that visits a monitored location at t. This time t will use to reset the clock. Secondly, the clock valuation depends on the TIS ρ, on the reference time of evaluation t, and on the *rate* τ. It is easy to see that for each $1 \le i \le n$ the clock valuation assigns a (non-standard) positive real, or undefined, to each clock variable $v(\rho, t, \tau)_{\mathscr{A}} = v^{'}(\rho, t, \tau)_{\mathsf{Det}(\mathscr{A})}$. Thirdly, since $s_{n-1} \in s^{'}_{n-1}$ and there exists a transition δ_i from (s_{n-1}, s_n) in $\mathscr{A}$, there exists $s^{'}_n$ such that $s_n \in s^{'}_n$ and there exists a transition $\delta^{'}_n$ from $(s^{'}_{n-1}, s^{'}_n)$ where the clock constraints are satisfied by the valuation $v^{'}(\rho, t, \tau)_{\mathsf{Det}(\mathscr{A})}$. Hence, we can add $s^{'}_{n-1}$, $\delta^{'}_n$, $s^{'}_n$ to the run over $\mathsf{Det}(\mathscr{A})$. It follows immediately that if $\mathscr{A}$ accepts the set of TIS ρ at time t and on the *rate* τ, then the language $\mathscr{L}^+_\exists(\mathscr{A}, \tau)$ is defined as the set of TIS ρ of accepting runs of $\mathscr{A}$ with regard to τ, and the language $\mathscr{L}^+_\exists(\mathsf{Det}(\mathscr{A}), \tau)$ is defined as the set of TIS ρ of accepting runs of $\mathsf{Det}(\mathscr{A})$ with regard to τ. We have that $\mathscr{L}^+_\exists(\mathscr{A}, \tau) = \mathscr{L}^+_\exists(\mathsf{Det}(\mathscr{A}), \tau)$. The other direction of the implication is proved using a similar argument. □

proposition 3. *For every DECA $\mathscr{A}$, Det($\mathscr{A}$) accepts the same universal language $\mathscr{L}_\forall(Det(\mathscr{A}), P) = \mathscr{L}_\forall(\mathscr{A}, P)$ for any $P \subseteq Proc$ then for all $\mathscr{A}$ and deterministic Det($\mathscr{A}$).*

Proof. We can derive this easily using 2. □

Theorem 37. *The τ-wise emptiness problem for DECA is PSPACE-complete.*

Proof. Using region technique [AD94] and [Ras99] we can show that the emptiness problem for RECA is decidable. As we have shown in this chapter, a RECA is a DECA. The emptiness problem of DECA is decidable in $n \cdot 2^{m \cdot \log(m \cdot c)}$, where n is the number of locations, m is the number of clocks, and c is the largest constant that appears in clock constraints, it follows that emptiness of a DECA $\mathscr{A}$ can be checked in PSPACE. □

Theorem 38. *The τ-wise language inclusion problem for DECA is PSPACE-complete.*

Proof. Consider the DECA $\mathscr{A}$ and $\mathscr{B}$ with regard to $\tau \in$ *rates*, such that each automaton has at most n locations, let m be the number of clocks. Let c be the largest constant that appears in the clock constraints. To check whether $\mathscr{L}(\mathscr{A},\tau) \subseteq \mathscr{L}(\mathscr{B},\tau)$, we first determinize $\mathscr{B}$ to $\mathscr{B}^1$ (Deterministic Rabin Automaton) used Theorem 5.2.1, after we translate $\mathscr{B}^1$ to a Muller Automaton and complement $\mathscr{B}^1$ to $\mathscr{B}^2$(complement Muller automaton). The automata $\mathscr{B}^2$ has $2^{m \cdot \log(m \cdot c)}$ locations since it is a ω automaton (Rabin automaton) and the integer constants that appear in the clock constraints of $\mathscr{B}^2$ are bounded by c. Let $\mathscr{D}$ be the intersection of $\mathscr{A}$ and $\mathscr{B}^2$. The DECA $\mathscr{D}$ has $n \cdot 2^{m \cdot \log(m \cdot c)}$ locations, where the integer constants that appear in the clock constraints of $\mathscr{D}$ are also bounded by c. □

Theorem 39. *The The τ-wise universality for DECA is PSPACE-Complete.*

Proof. We use the model of [AL99] to prove by reducing the acceptance of a word w by a Linear Bounded Alternating Turing Machine (LBATM) $\mathscr{M}$ to a model checking problem for DECA. One can assume that the alphabet of $\mathscr{M}$ is $\{a,b\}$, and let $n = |w|$.

Let $\mathscr{M} = \langle Q, \Sigma, q_0, q_f, \rightarrow_{tm}\rangle$, where Σ is the alphabet, Q is the set of states that is partitioned into Q_{or} and Q_{and}, $\rightarrow \subseteq Q \times \Sigma \times \Sigma \times \{L,R\} \times Q$ is the transition relation (we use $(q,\sigma,\sigma',\delta,q')$ to indicate that when $\mathscr{M}$ is in state q and it reads the input σ in the current tape cell and the head over it, writes σ' in the current tape cell, and its reading head moves one cell to the left, or to the right, according to δ and it moves to state q'), q_0 is the initial state, and q_f is the final state.

A configuration of $\mathscr{M}$ is a triple $(q,w,i) \in Q \times \Sigma^* \times \mathbb{N}$ where q is a control location, $w \in \Sigma^*$ is the content of the tape, and $i \leq n$ is the position of the tape head. A configuration (q',w',i') is a successor of a configuration (q,w,i) iff there exists a transition $(q,\sigma,\sigma',\delta,q') \in \rightarrow_{tm}$ such that:

(i) $w_i = \sigma$,

(ii) $w'_i = \sigma'$ and $w'_j = w_j$ for all $j \neq i$,

(iii) $i' = i-1$ if $\delta = L$ and $i' = i+1$ if $\delta = R$ with $1 \leq i' \leq |w|$.

We assume that the condition $1 \leq i' \leq |w|$ is realized using input delimiters. An execution of $\mathscr{M}$ on the input $w \in \Sigma^*$ is a sequence $s_0 s_1 \cdots s_n$ of configurations starting with $s_0 = (q_0, w, 1)$ and such that s_{i+1} is a successor of s_i for every $0 \leq i < n$. We say that $\mathscr{M}$ accepts w iff $\mathscr{M}$ has an execution on w finishing in $s_n = (q_f, w, i)$ for some $w \in \Sigma^*$ and $i \in \mathbb{N}$. The acceptance problem for LBATM asks, given a LBATM $\mathscr{M}$ and an input word $w \in \sigma^*$ whether $\mathscr{M}$ accepts w.

We want to build an DECA $\mathscr{A}_{\mathscr{M},w}$, while the construction of [AL99] is done for TA. We establish PSPACE reduction using the acceptance problem for $\mathscr{M}$ which is known to be PSPACE-complete [CKS81]. Our reduction is similar to [AL99], where a configuration (q, w, i) of a $\mathscr{M}$ is encoded by a location (q, i) (that records the control state q and the tape position i) and by the two clocks x_i^p, y_i^p for each tape cell and $p \in Proc$, where $1 \leq i \leq |w|$. The cell i of the tape contains an a when the constraints $x_i^p = y_i^p$ holds, and cell i contains a b when the constraint $x_i^p < y_i^p$ holds. This two conditions are invariant by time elapsing. To force time elapsing between two transitions corresponding to moves of $\mathscr{M}$, this can be done by adding a clock z^p for each process $p \in Proc$, which is checked to 1 and reset to 0 on all transitions. If $(q, \sigma, \sigma', \delta, q')$ is a transition of the $\mathscr{M}$, then for each position i of the tape, it will be represented in $\mathscr{A}$ by the transitions $(q, i) \xrightarrow{\phi_i, Y \to 0} (q', i')$, where:

(i) ϕ_i is $x_i^p = y_i^p \wedge z^p = 1$ if $\sigma = a$ and ϕ_i is $x_i^p < y_i^p \wedge z^p = 1$ if $\sigma = b$,
(ii) $Y = \{x_i^p, y_i^p\}$ if $\sigma = a$ and $Y = \{x_i^p\}$ if $\sigma = b$,
(iii) $i' = i + 1$ if $\delta = R$ and $i < n$, and $i' = i - 1$ if $\delta = L$ and $i > 1$.

Initially the tape contains the encoding of the word w. This can be done by a transition from a initial state $(q_0, 1)$ where q_0 is the initial state of the $\mathscr{M}$, which checks whether $z^p = 1$, and resets clocks in Y_0 where $Y_0 = \{z^p\} \cup \{x_i^p | w_i = b \; and \; p \in Proc\}$. We distinguish three labels for actions of $\mathscr{M}$: $Init$ corresponds to the writing of w on the tape at the beginning of the computation, a labels any step of $\mathscr{M}$ and $Accepting$ labels accepting states of $\mathscr{M}$. The computation over w of the $\mathscr{M}$ terminates iff there is a run from initial state to some state (q_f, i) where q_f is the final state of the $\mathscr{M}$. □

From Theorems 37, 38 and 39, we have the next Theorem 40.

Theorem 40. *The existential emptiness and language inclusion problems for DECA are PSPACE-complete.*

Proof. We can derive this easily using the Theorem 37 and 38. □

Theorem 41. *The universal emptiness and language inclusion problems for DECA are PSPACE-complete.*

Proof. We can derive this easily using the Theorem 38 and 39. □

proposition 4. *For all DECA $\mathscr{B}, Q \subseteq Proc, \mathscr{L}_\exists(\mathscr{B}, Q)$ is the language of a DECA.*

Proof. For the existential languages of DECA, we can eliminate a process q from a DECA while preserving the existential language of the remaining processes. We first complete and determinize automata appearing in the clocks of this process q. We then make their product with the main automaton. We then perform the region construction [AFH96] on the clocks of q. Remember that the clocks are constrained to be 0 in the respective monitored location, i.e. when at least one original monitored location appears in this construction, and that prediction clocks run backwards so that it is the complement of their fractional part that participates in the region construction [AFH96]. The region construction for prediction clocks is non-deterministic and is not a bisimulation quotient, unlike the one of TA, but preserves the language [Ras99]. Note that the elimination of the clocks of one process only, allows independent evolution of the other clocks. The resulting automaton is still a DECA. □

proposition 5. *For all DECA $\mathscr{B}, q \in Proc, \mathscr{L}_\forall(\mathscr{B}, \{q\})$ is the language of a RECA.*

Proof. For the universal languages $\mathscr{L}_\forall(\mathscr{B}, Q)$ of a DECA $\mathscr{B}$, we complete and determinize the main automaton $\mathscr{B}$. Then we apply the region construction for independent clocks [ABG$^+$08]. The automaton becomes non-deterministic, because each region has several successors, depending on τ. Note that the region construction for ECA was already non-deterministic. A region constraint is expressed as a conjunction $\bigwedge_{p \in Proc} \phi_p$. We label each region state by $\bigwedge_{p \in Q} \phi_p$. Then we determinize it again but we mark as final the locations where all members are final (which, in turn, means that one of their members is an original final state), to represent that the ISS must be accepted under all evolutions of time τ. The resulting automaton is a RECA. □

proposition 6. *For all DECA $\mathscr{B}, q \in Proc, \mathscr{L}_\forall(\mathscr{B}, \{q\})$ is the language of a DECA.*

Proof. For the universal languages of DECA, we can eliminate a process q from a DECA while preserving the universal language of the remaining processes. We first complete and determinize automata appearing in the clocks of this process q. We then make their product with the main automaton. We then perform the region construction [AFH96] on the clocks of q. Remember that the clocks are constrained to be 0 in the respective monitored location, i.e. when at least one original monitored location appears in this construction, and that prediction clocks run backwards so that it is the complement of their fractional part that participates in the region construction [AFH96]. The region construction for prediction clocks is non-deterministic and is not a bisimulation quotient, unlike the one of TA, but preserves the language [Ras99]. Note that the elimination of the clocks of one process only, allows independent evolution of the other clocks. The resulting automaton is still a DECA. □

proposition 7. *For all DECA $\mathscr{B}, Q \subseteq Proc, \mathscr{L}_\exists(\mathscr{B}, Q)$ is the language of a RECA.*

Proof. For the existential languages $\mathscr{L}_\exists(\mathscr{B}, Q)$ of a DECA $\mathscr{B}$, we complete and determinize the main automaton $\mathscr{B}$. Then we apply the region construction for independent clocks [ABG$^+$08]. The automaton becomes non-deterministic, because each region has several successors, depending on τ. Note that the region construction for ECA was already non-deterministic. A region constraint is expressed as a conjunction $\bigwedge_{p \in Proc} \phi_p$. We label each region state by $\bigwedge_{p \in Q} \phi_p$. Then we determinize it again but we mark as final the locations where all members are final (which, in turn, means that one of their members is an original final state), to represent that the ISS must be accepted under all evolutions of time τ. The resulting automaton is a RECA. □

6.1.6 Region Construction DECA

The principles of the called region automaton which transforms a timed automaton $\mathscr{A}$ (RG($\mathscr{A}$)) into an untimed finite automaton can be applied to DECA [AFH94] [Ras99]. The following theorem is the basis for an algorithmic analysis of DECA automata:

Theorem 42. *The number of locations in the region automaton RG($\mathscr{A}$) of a DECA automata $\mathscr{A}$ is $n \cdot 2^{O(m \cdot \log(m \cdot c)}$, where n is the number of locations in $\mathscr{A}$, m is the number of clocks and c is the largest constant appearing in $\mathscr{A}$.*

Proof. Let $\mathscr{M}$ be a RECA, where n is the number of locations, m is the number of clocks, and c is the largest integer constant that appears in the clock constraints. From [Ras99], it follows that the number of locations in the region automaton is $n \cdot 2^n \cdot 2^{O(m \cdot \log(m \cdot c))}$. The inclusion problem for RECA is in fact PSPACE-complete. In case of DECA the number of clocks are as in RECA $n \cdot 2^n \cdot 2^{O(m \cdot \log(m \cdot c)}$. □

Theorem 43. *For every run on the given DECA $\mathscr{A}$, there is an untimed run on the region automaton of $\mathscr{R}(\mathscr{A})$. The region automaton precisely accepts the language Untimed($\mathscr{L}(\mathscr{A}, \tau)$).*

Proof. The language emptiness is decidable for RG($\mathscr{A}$), since it is a finite state automaton. One can easily see that RG($\mathscr{A}$) can be effectively constructed and that $\mathscr{L}(\mathscr{A}, \tau)$ is empty iff Untimed($\mathscr{L}(\mathscr{A}, \tau)$) is empty. □

In the following section we present a new concept of timed languages for DTA and icTA.

6.2 Multi-Timed Languages for DTA and icTA

Surprisingly, Akshay et al. [ABG$^+$08] only consider untimed languages for their timed automata (DTA and icTA). We are interested in timed languages, but we have several times here. We thus define a multi-TIS as a sequence of letters and (local time) interval sequences, one for each local time in $P \subseteq \mathit{Proc}$: $\mu = (\sigma, (I_q)_{q \in P})$. Let τ_P be a rate defined on P. Given an interval I, we can obtain the corresponding multi-interval $\tau_P(I)$ by applying τ_P to its bounds, for instance $\tau_P(]t_i, t_{i+1}[) =]\tau_P(t_i), \tau_P(t_{i+1})[$. This extends naturally to interval sequences. Given an TIS $\rho = (\sigma, I)$ (expressed in the reference time), its multi-TIS $\tau_P(\rho)$ is $(\sigma, \tau_P(I))$. Let $\mathscr{B}$ be an icTA, the language $\mathscr{L}(\mathscr{B}, \tau)$, is defined as the set of TIS of accepting runs of $\mathscr{B}$ for τ, closed under the equivalence generated by merging adjacent intervals with the same propositional labeling. However, TIS is not significant here, since it is expressed in the (arbitrary) reference time. The multi-timed language $\mathscr{L}(\mathscr{B}, \tau, P)$ is the set of all accepted multi-TIS: $\mathscr{L}(\mathscr{B}, \tau, P) = \tau_P(\mathscr{L}(\mathscr{B}, \tau))$. If we select a subset Q of P, we can project a multi-TIS $\rho = (\sigma, (I_q)_{q \in P})$ to this subset Q, noted $\rho|_Q = (\sigma, (I_q)_{q \in Q})$. This projection extends naturally to languages. In particular, the UNTIME operation [AD94] is the case with $P = \emptyset$. Note that $\mathscr{L}(\mathscr{B}, \tau, P)|_Q = \mathscr{L}(\mathscr{B}, \tau, Q)$. When there is only one process $\mathit{Proc} = \{q\}$, the timed language observed by q, $\mathscr{L}((\mathscr{A}, \pi), \tau, \{q\})$, does not depend on τ, and for an icTA, it is the usual timed language $\mathscr{L}(\mathscr{A})$ of its TA. When τ is the identity, we also obtain the usual timed language.

For expressing real-time requirements, we have to choose the process(es) that will measure the time. When we want to avoid some forbidden timed behaviors, we naturally consult the existential timed semantics. We consider local times as non-deterministic. If we want a given timed behavior to be possible whatever the evolution of local times, we check that it belongs to the universal semantics. Thus we define, for an automaton $\mathscr{B}$ and a subset of its processes P:

- The existential timed language observed by $P : \mathscr{L}_\exists(\mathscr{B},P) = \bigcup_{\tau\in Rates} \mathscr{L}(\mathscr{B},\tau,P)$
- The universal timed language observed by $P : \mathscr{L}_\forall(\mathscr{B},P) = \bigcap_{\tau\in Rates} \mathscr{L}(\mathscr{B},\tau,P)$

The untimed languages defined in [ABG$^+$08] are special cases with an empty set of observers. More generally:

Theorem 44. *$\mathscr{L}_\exists(\mathscr{B},P)|_Q = \mathscr{L}_\exists(\mathscr{B},Q)$ where $Q \subseteq P$.*

Proof.

$$\begin{aligned}
\mathscr{L}_\exists(\mathscr{B},P)|_Q &= (\bigcup_{\tau\in Rates} \mathscr{L}(\mathscr{B},\tau,P))|_Q \\
&= (\bigcup_{\tau\in Rates} \tau_P(\mathscr{L}(\mathscr{B},\tau)))|_Q \\
&= (\bigcup_{\tau\in Rates} \{\tau_P(\sigma,I) \mid (\sigma,I)\in\mathscr{L}(\mathscr{B},\tau))\})|_Q \\
&= (\{(\sigma,\tau_P(I)) \mid (\sigma,I)\in\mathscr{L}(\mathscr{B},\tau)) \wedge \tau\in Rates\})|_Q \\
&= \{(\sigma,\tau_Q(I)) \mid (\sigma,I)\in\mathscr{L}(\mathscr{B},\tau)) \wedge \tau\in Rates\} \\
&= \mathscr{L}_\exists(\mathscr{B},Q)
\end{aligned}$$

□

But for universal languages, we need to prove the following theorems:

Theorem 45. *For any icTA $\mathscr{B}$, if $|P| \geq 2$ then $\mathscr{L}_\forall(\mathscr{B},P) = \emptyset$ where $P \subseteq Proc$.*

Proof. We prove this by contradiction. Consider the set of processes $P = \{p_1, p_2, \ldots, p_n\}$ with $n \geq 2$, the universal timed language $\mathscr{L}_\forall(\mathscr{B},P)$ where $P \subseteq Proc$ and an TIS ρ, then we want to prove that for all $\tau \in Rates$, $\tau_P(\rho) \notin \mathscr{L}(\mathscr{B},\tau,P)$. Assume the contrary that for some $P = \{p_1, p_2, \ldots, p_n\}$ with $n \geq 2$, the universal timed language $\mathscr{L}_\forall(\mathscr{B},P) = \bigcap_{\tau\in Rates}\mathscr{L}(\mathscr{B},\tau,P) = \bigcap_{\tau\in Rates}\tau_P(\mathscr{L}(\mathscr{B},\tau)) \neq \emptyset$, then there is ρ, for all $\tau \in Rates$, $\tau_P(\rho) \in \mathscr{L}(\mathscr{B},\tau,P)$. Consider the TIS $\rho = (\gamma(\zeta_0),\{0\}), (\gamma(s_1),]0, t_1[), \ldots$, the tuple of local functions $\tau = (\tau_{p_1}, \tau_{p_2}, \ldots, \tau_{p_n})$ and since $\neg\forall\tau$, $\tau_P(\rho) \in \mathscr{L}(\mathscr{B},\tau,P)$, then we have that $\neg\forall\tau, \exists\theta$ that is $(s_0,v_0,t_0) \xrightarrow{\zeta_0} (s_1,v_1,t_1)\ldots$ a run of $\mathscr{B}$, where $\tau_P(\rho) = \tau_P(\mathsf{TIS}(\theta))$. Given $\mathscr{B}$, $\tau \in Rates$ and $\rho = (\gamma(\zeta_0),\{0\}), (\gamma(s_1),]0, t_1[), \ldots$, we have that $\tau_{p_1}(t_1) = t_{p_1}$ with $\kappa_1 \leq t_{p_1} \leq \kappa_2$, $\tau_{p_2}(t_1) = t_{p_2}$ with $\kappa_1 \leq t_{p_2} \leq \kappa_2$, ..., $\tau_{p_n}(t_1) = t_{p_n}$ with $\kappa_1 \leq t_{p_n} \leq \kappa_2$, where we assume due to the construction that κ_1, κ_2 are both integers and for each $1 \leq i \leq n$, $t_{p_i} \in [\kappa_1,\kappa_2]$. Let $\tau_{p_1}(t) = t_{p_1}\cdot t$, $\tau_{p_2}(t) = 2\cdot t_{p_2}\cdot t$, ..., $\tau_{p_n}(t) = n\cdot t_{p_n}\cdot t$ implies $t_{p_1}\cdot t_1 = t_{p_1}$ then $t_1 = 1$, $2\cdot t_{p_2}\cdot t_1 = t_{p_2}$ then $t_1 = 1/2, \ldots$, $n\cdot t_{p_n}\cdot t_1 = t_{p_n}$ then $t_1 = 1/n$ which is impossible and contradicts that $\mathscr{L}_\forall(\mathscr{B},P) \neq \emptyset$. □

Theorem 46. *For some icTA $\mathscr{B}$ with final states $F \neq \emptyset$, if $|P| < 2$ then $\mathscr{L}_\forall(\mathscr{B}, P) \neq \emptyset$ where $P \subseteq Proc$.*

Proof. We prove this by contradiction. Consider the set of a single process $P = \{p\}$ and the universal timed language $\mathscr{L}_\forall(\mathscr{B}, P)$ where $P \subseteq Proc$ and an TIS ρ, then we want to prove that for all $\tau \in Rates$, $\tau_P(\rho) \in \mathscr{L}(\mathscr{B}, \tau, P)$. Assume the contrary that for some $P = \{p\}$, the universal timed language $\mathscr{L}_\forall(\mathscr{B}, P) = \bigcap_{\tau \in Rates} \mathscr{L}(\mathscr{B}, \tau, P)$ $= \bigcap_{\tau \in Rates} \tau_P(\mathscr{L}(\mathscr{B}, \tau)) = \emptyset$, then there is ρ, for at least one $\tau \in Rates$, $\tau_P(\rho) \notin \mathscr{L}(\mathscr{B}, \tau, P)$. Consider the TIS $\rho = (\gamma(\zeta_0), \{0\}), (\gamma(s_1),]0, t_1[), \ldots$, the tuple of local functions $\tau = (\tau_p)$ and since $\neg\forall\tau$, $\tau_P(\rho) \in \mathscr{L}(\mathscr{B}, \tau, P)$, then we have that $\neg\forall\tau$, $\exists\theta$ that is $(s_0, v_0, t_0) \xrightarrow{\zeta_0} (s_1, v_1, t_1) \ldots$ a run of $\mathscr{B}$, where $\tau_P(\rho) = \tau_P(\mathsf{TIS}(\theta))$. Given $\mathscr{B}$, $\tau \in Rates$ and $\rho = (\gamma(\zeta_0), \{0\}), (\gamma(s_1),]0, t_1[), \ldots$, we have that $\tau_{p_1}(t_1) = t_p$ with $\kappa_1 \leq t_{p_1} \leq \kappa_2$, where we assume due to the construction that κ_1, κ_2 are both integers and $t_p \in [\kappa_1, \kappa_2]$. Let $\tau_p(t) = t_p \cdot t$, implies $t_p \cdot t_1 = t_p$ then $t_1 = 1$, which contradicts that $\mathscr{L}_\forall(\mathscr{B}, P) = \emptyset$. □

Theorem 47. *$\mathscr{L}_\forall(\mathscr{B}, P)|_Q \subseteq \mathscr{L}_\forall(\mathscr{B}, Q)$ where $Q \subseteq P$.*

Proof. Consider the set of processes $P = \{p_1, p_2, \ldots, p_n\}$ with $n \geq 2$, the universal timed language $\mathscr{L}_\forall(\mathscr{B}, P)|_Q$ where $Q \subseteq P \subseteq Proc$ and $2 \leq |Q| \leq n$, then we want to prove that $\mathscr{L}_\forall(\mathscr{B}, P)|_Q \subseteq \mathscr{L}_\forall(\mathscr{B}, Q)$.

$$\begin{aligned} \mathscr{L}_\forall(\mathscr{B}, P)|_Q &= (\bigcap_{\tau \in Rates} \mathscr{L}(\mathscr{B}, \tau, P))|_Q \\ &= \emptyset \subseteq \mathscr{L}_\forall(\mathscr{B}, Q) \end{aligned}$$

□

Theorem 48. *For any icTA $\mathscr{B}$, $\mathscr{L}_\exists(\mathscr{B}, Q)$ is the language of an icTA on Q.*

Proof. Existential timed languages can be computed by a variant of the region construction, of which the construction of [ABG$^+$08] is a special case. Let $q \in Proc$ be a process whose clocks we want to eliminate, i.e. we have an icTA $\mathscr{B}$ on *Proc* and we would like to construct an icTA on $Proc \setminus \{q\}$ whose existential language is $\mathscr{L}_\exists(\mathscr{B}, Proc \setminus \{q\})$. We construct the region equivalence, but on the clocks of q only. This gives a region icTA without the clocks of q, and where the locations are now a pair of an original location and a region constraint on clocks of q, which has the required language. If we want to eliminate several processes, we eliminate them one by one: eliminating several processes together would give a result that does not reflect the independence of their clocks. □

Theorem 49. *For any icTA $\mathscr{B}$, $\mathscr{L}_\exists(\mathscr{B}, \{q\})$ is the language of a TA.*

Proof. Existential timed language can be computed by the region construction [ABG$^+$08] of an icTA for the independent clocks of the single process $q \in Proc$, whose clocks evolve at the same speed, i.e., they follow the same clock rate, then our model corresponds to a standard timed automata [AD94]. We construct the region equivalence with the clocks of q. A region constraint is of the form $\bigwedge_{q\in Proc}\phi_q$. This gives a region icTA with the clocks of q, and where the locations are now a pair of an original location and a region constraint on clocks of q, which has the required language. We label the region state by $\bigwedge_{q\in Proc}\phi_q$. Then we mark as final the locations where all members are final (which, in turn, means that one of their members is an original final state), to represent that the ISS must be accepted under evolution of time τ_q. The resulting automaton is a TA. □

This variety of languages leads to three generalisations of the classical problems of emptiness, inclusion, intersection and union. First, the τ-wise definitions:

Definition 65. *Given icTA $\mathscr{A},\mathscr{B},\mathscr{C}$,*

(i) $\mathscr{C}$ is a τ-intersection of $\mathscr{A},\mathscr{B}$ iff $\forall\tau\in Rates, \mathscr{L}(\mathscr{C},\tau)=\mathscr{L}(\mathscr{A},\tau)\cap\mathscr{L}(\mathscr{B},\tau)$

(ii) $\mathscr{C}$ is a τ-union of $\mathscr{A},\mathscr{B}$ iff $\forall\tau\in Rates, \mathscr{L}(\mathscr{C},\tau)=\mathscr{L}(\mathscr{A},\tau)\cup\mathscr{L}(\mathscr{B},\tau)$

(iii) $\mathscr{C}$ is a τ-complement automaton of $\mathscr{A}$ iff $\forall\tau\in Rates, \mathscr{L}(\mathscr{C},\tau)=\mathscr{L}(\mathscr{A},\tau)^c$, where c is the complement operator.

(iv) $\mathscr{A}$ is a τ-language-included in $\mathscr{B}$ iff $\forall\tau\in Rates, \mathscr{L}(\mathscr{A},\tau)\subseteq\mathscr{L}(\mathscr{B},\tau)$

(v) The τ-emptiness problem for $\mathscr{A}$ is $\forall\tau\in Rates, \mathscr{L}(\mathscr{A},\tau)=\emptyset$

The existential use respectively the existential timed language observed by $P\subseteq$ *Proc*.

Definition 66. *Given icTA $\mathscr{A},\mathscr{B},\mathscr{C}$,*

(i) $\mathscr{C}$ is an $\exists$-intersection observed by P of $\mathscr{A},\mathscr{B}$ iff $\mathscr{L}_\exists(\mathscr{C},P)=\mathscr{L}_\exists(\mathscr{A},P)\cap\mathscr{L}_\exists(\mathscr{B},P)$

(ii) $\mathscr{C}$ is an $\exists$-union observed by P of $\mathscr{A},\mathscr{B}$ iff $\mathscr{L}_\exists(\mathscr{C},P)=\mathscr{L}_\exists(\mathscr{A},P)\cup\mathscr{L}_\exists(\mathscr{B},P)$

(iii) $\mathscr{C}$ is an $\exists$-complement automaton observed by P of $\mathscr{A}$ iff $\mathscr{L}_\exists(\mathscr{C},P)=\mathscr{L}_\exists(\mathscr{A},P)^c$, where c is the complement operator.

(iv) $\mathscr{A}$ is $\exists$-language-included observed by P in $\mathscr{B}$ iff $\mathscr{L}_\exists(\mathscr{A},P)\subseteq\mathscr{L}_\exists(\mathscr{B},P)$

(v) The $\exists$-emptiness problem observed by P for $\mathscr{A}$ is $\mathscr{L}_\exists(\mathscr{A},P)=\emptyset$

The universal use respectively the universal timed language observed by $P\subseteq$ *Proc*.

Definition 67. *Given icTA $\mathscr{A},\mathscr{B},\mathscr{C}$,*

(i) $\mathscr{C}$ is a $\forall$-intersection observed by P of $\mathscr{A},\mathscr{B}$ iff $\mathscr{L}_\forall(\mathscr{C},P)=\mathscr{L}_\forall(\mathscr{A},P)\cap\mathscr{L}_\forall(\mathscr{B},P)$

(ii) $\mathscr{C}$ is a $\forall$-union observed by P of $\mathscr{A},\mathscr{B}$ iff $\mathscr{L}_\forall(\mathscr{C},P)=\mathscr{L}_\forall(\mathscr{A},P)\cup\mathscr{L}_\forall(\mathscr{B},P)$

(iii) $\mathscr{C}$ is a $\forall$-complement automaton observed by P of $\mathscr{A}$ iff $\mathscr{L}_\forall(\mathscr{C},P)=\mathscr{L}_\forall(\mathscr{A},P)^c$, where c is the complement operator.

(iv) $\mathscr{A}$ is $\forall$-language-included observed by P in $\mathscr{B}$ iff $\mathscr{L}_\forall(\mathscr{A},P)\subseteq\mathscr{L}_\forall(\mathscr{B},P)$

(v) The $\forall$-emptiness problem observed by P for $\mathscr{A}$ is $\mathscr{L}_\forall(\mathscr{A},P)=\emptyset$

The τ-wise definitions are indeed the strongest:

Theorem 50. *icTA is closed under union operation.*

Proof. Let $\mathscr{A}=(S^{\mathscr{A}},s_0^{\mathscr{A}},\Sigma^{\mathscr{A}},X^{\mathscr{A}},\gamma^{\mathscr{A}},Inv^{\mathscr{A}},\rightarrow_{icta}^{\mathscr{A}},F^{\mathscr{A}},\pi^{\mathscr{A}})$ and $\mathscr{B}=(S^{\mathscr{B}},s_0^{\mathscr{B}},\Sigma^{\mathscr{B}},X^{\mathscr{B}},\gamma^{\mathscr{B}},Inv^{\mathscr{B}},\rightarrow_{icta}^{\mathscr{B}},F^{\mathscr{B}},\pi^{\mathscr{B}})$ be two icTA. Without loss of generality we assume that the sets of clocks $X^{\mathscr{A}}$ and $X^{\mathscr{B}}$ (and respectively the sets of locations $S^{\mathscr{A}}$ and $S^{\mathscr{B}}$) are all pairwise disjoint. Let $\mathscr{C} = (\Sigma^{\mathscr{C}}, X^{\mathscr{C}}, S^{\mathscr{C}}, s_0^{\mathscr{C}}, \rightarrow_{icta}^{\mathscr{C}}, \gamma^{\mathscr{C}}, Inv^{\mathscr{C}}, F^{\mathscr{C}}, \pi^{\mathscr{C}})$ be the icTA defined as follows:

(i) $S^{\mathscr{C}} = S^{\mathscr{A}} \cup S^{\mathscr{B}}$,
(ii) $s_0^{\mathscr{C}} = (s_0^{\mathscr{A}}, s_0^{\mathscr{B}})$,
(iii) $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} \cup \Sigma^{\mathscr{B}}$,
(iv) $X^{\mathscr{C}} = X^{\mathscr{A}} \cup X^{\mathscr{B}}$,
(v) $\gamma^{\mathscr{C}} = \gamma^{\mathscr{A}} \cup \gamma^{\mathscr{B}}$,
(vi) $Inv^{\mathscr{C}} = Inv^{\mathscr{A}} \cup Inv^{\mathscr{B}}$,
(vii) $\rightarrow_{icta}^{\mathscr{C}} = \rightarrow_{icta}^{\mathscr{A}} \cup \rightarrow_{icta}^{\mathscr{B}}$,
(viii) $F^{\mathscr{C}} = F^{\mathscr{A}} \cup F^{\mathscr{B}}$,
(ix) $\pi^{\mathscr{C}} = \pi^{\mathscr{A}} \cup \pi^{\mathscr{B}}$,

The proof of union correctness is the following: We need to show that for all $\tau \in Rates$, $\rho \in \mathscr{L}(\mathscr{C}, \tau)$ iff $\rho \in \mathscr{L}(\mathscr{A}, \tau) \cup \mathscr{L}(\mathscr{B}, \tau)$. Assume that $\rho = (\gamma(\zeta_0), \{t_0\}), (\gamma(s_1),]t_1, t_2[), \ldots, (\gamma(s_n),]t_{n-1}, t_n[) \in \mathscr{L}(\mathscr{C}, \tau)$ is an TIS and an icTA $\mathscr{C}$. Then ρ originates from some runs $\theta = (s_0^{\mathscr{C}}, \nu_0, t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{C}}, \nu_1, t_1) \ldots \xrightarrow{\zeta_{n-1}} (s_n^{\mathscr{C}}, \nu_n, t_n)$ of $\mathscr{C}$ with regard to τ, where a run is an alternating sequence of states and transitions. The states are triples of a location, a clock valuation, and lastly the reference time: $\{(s, \nu, t) \in S^{\mathscr{C}} \times \mathbb{R}_{\geq 0}^{X} \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s)\}$. The transitions must alternate between two types: (i) Delay transition, i.e. spending time in a location: $q_i \xrightarrow{d} q_i + d$, where $q_i = (s_i, \nu_i, t_i)$, and $q_i + d = (s_i, \nu_i + (\tau(t_i + d) - \tau(t_i)), t_i + d)$, if the invariant is continuously true in local time: $\forall t \in]t_i, t_i + d[: \nu_i + (\tau(t) - \tau(t_i)) \models Inv(s_i)$. (ii) Discrete transition: following a transition $\zeta_i = (s_{i-1}, \phi, Y, s_i) \in \rightarrow_{\text{icTA}}$ when the clock constraint ϕ is satisfied. The clocks in Y are then reset. This transition is instantaneous. $(s_{i-1}, \nu_{i-1}, t_{i-1}) \xrightarrow{\zeta_i} (s_i, \nu_i, t_i)$, such that $\nu_{i-1} \models \phi$, $\nu_i = \nu_{i-1}[Y \rightarrow 0]$, $t_{i-1} = t_i$. For any clock $x^{\mathscr{C}} \in X^{\mathscr{C}}$, $\nu(x^{\mathscr{C}}) = \tau(t) - \tau(t_i)$ where $i \geq 0$ is the index of the last transition which reset clock $x^{\mathscr{C}}$ or is $\tau(t)$ if $x^{\mathscr{C}}$ was never reset. We can say the $\mathscr{C}$ accepts ρ at time t with regard to τ, if there is a run θ for an equivalent of ρ that visits an accepting location $s_n^{\mathscr{C}} \in F^{\mathscr{C}}$ at t and on the *rate* τ, where we can safely assume due to the construction that, for each $1 \leq i \leq n$, $s_0^{\mathscr{C}} \xrightarrow{\rho}_{\mathscr{C}} s_n^{\mathscr{C}}$. By induction hypothesis we know that $\rho = (\gamma(\zeta_0), \{t_0\}), (\gamma(s_1),]t_1, t_2[), \ldots, (\gamma(s_n),]t_{n-1}, t_n[) \in (\mathscr{L}(\mathscr{A}, \tau) \cup \mathscr{L}(\mathscr{B}, \tau))$, then there is an accepting run $\theta' = (s_0^{\mathscr{A} \cup \mathscr{B}}, \nu_0', t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{A} \cup \mathscr{B}}, \nu_1', t_1) \ldots \xrightarrow{\zeta_{n-1}} (s_n^{\mathscr{A} \cup \mathscr{B}}, \nu_n', t_n)$ be the run for ρ that visits an accepting location $s_n^{\mathscr{A} \cup \mathscr{B}} \in (F^{\mathscr{A}} \cup F^{\mathscr{B}})$ at t and on the *rate* τ, where we can safely assume due to the construction that, for each $1 \leq i \leq n$, $s_0^{\mathscr{A} \cup \mathscr{B}} \xrightarrow{\rho}_{\mathscr{A} \cup \mathscr{B}} s_n^{\mathscr{A} \cup \mathscr{B}}$. Firstly, we say that $\mathscr{C}$ accepts an TIS ρ at t with regard to *rate* τ, if there is a run θ for ρ that visits a accepting location $s_n^{\mathscr{C}} \in F^{\mathscr{C}}$ at t. Secondly the clock valuation depends on the TIS ρ, on the reference time of evaluation t, and on the *rate* τ. It is easy to see that for each $1 \leq i \leq n$ the clock valuation assigns a (non-standard) positive real to each clock variable $\nu_i = \nu_i'$. Thirdly, since $s_{n-1}^{\mathscr{C}} \in (S^{\mathscr{A}} \cup S^{\mathscr{B}})$ and there exists a transition $\zeta_{n-1}^{\mathscr{C}} = (s_{n-2}^{\mathscr{C}}, \phi, Y, s_{n-1}^{\mathscr{C}}) \in \rightarrow_{icTA}^{\mathscr{C}}$, there exists $s_n^{\mathscr{A} \cup \mathscr{B}} \in (F^{\mathscr{A}} \cup F^{\mathscr{B}})$ such that $s_n^{\mathscr{C}} \in (F^{\mathscr{A}} \cup F^{\mathscr{B}})$ and there exists a transition $\zeta_n^{\mathscr{A} \cup \mathscr{B}} = ((s_{n-1}^{\mathscr{A}}, \phi, Y, s_n^{\mathscr{A}}) \in \rightarrow_{icTA}^{\mathscr{A}} \cup (s_{n-1}^{\mathscr{B}}, \phi, Y, s_n^{\mathscr{B}}) \in \rightarrow_{icTA}^{\mathscr{B}})$ where the clock constraints are satisfied by the valuation ν_n'. Hence we can adjunct $s_0^{\mathscr{A} \cup \mathscr{B}} \xrightarrow{\rho} s_n^{\mathscr{A} \cup \mathscr{B}}$ to the run over $\mathscr{A}$ and $\mathscr{B}$. It follows immediately that $\mathscr{C}$ accepts the set of TIS ρ at time t and on the *rate* τ, then the language $\mathscr{L}(\mathscr{C},\tau)$ is the set of TIS ρ of accepting runs of $\mathscr{C}$ with regard to τ, and the language $\mathscr{L}(\mathscr{A},\tau) \cup \mathscr{L}(\mathscr{B},\tau)$ is defined as the set of ISS ρ of accepting runs of the union of $\mathscr{A}$ and $\mathscr{B}$ with regard to τ. We have that $\mathscr{L}(\mathscr{C},\tau) = \mathscr{L}(\mathscr{A},\tau) \cup \mathscr{L}(\mathscr{B},\tau)$. □

Theorem 51. *icTA is closed under intersection operation.*

Proof. Let $\mathscr{A} = (S^{\mathscr{A}}, s_0^{\mathscr{A}}, \Sigma^{\mathscr{A}}, X^{\mathscr{A}}, \gamma^{\mathscr{A}}, Inv^{\mathscr{A}}, \rightarrow_{icta}^{\mathscr{A}}, F^{\mathscr{A}}, \pi^{\mathscr{A}})$ and $\mathscr{B} = (S^{\mathscr{B}}, s_0^{\mathscr{B}}, \Sigma^{\mathscr{B}}, X^{\mathscr{B}}, \rightarrow_{icta}^{\mathscr{B}}, \gamma^{\mathscr{B}}, Inv^{\mathscr{B}}, F^{\mathscr{B}}, \pi^{\mathscr{B}})$ be two icTA. Without loss of generality we assume that the sets of clocks $X^{\mathscr{A}}$ and $X^{\mathscr{B}}$ (and respectively the sets of locations $S^{\mathscr{A}}$ and $S^{\mathscr{B}}$) are all pairwise disjoint. Let $\mathscr{C} = (\Sigma^{\mathscr{C}}, X^{\mathscr{C}}, S^{\mathscr{C}}, s_0^{\mathscr{C}}, \rightarrow_{icta}^{\mathscr{C}}, \gamma^{\mathscr{C}}, Inv^{\mathscr{C}}, F^{\mathscr{C}}, \pi^{\mathscr{C}})$ be the icTA defined as follows:

(i) $S^{\mathscr{C}} = S^{\mathscr{A}} \times S^{\mathscr{B}}$,

(ii) $s_0^{\mathscr{C}} = (s_0^{\mathscr{A}}, s_0^{\mathscr{B}})$,

(iii) $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} \cap \Sigma^{\mathscr{B}}$,

(iv) $X^{\mathscr{C}} = X^{\mathscr{A}} \cup X^{\mathscr{B}}$,

(v) For all $s_1^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_2^{\mathscr{B}} \in S^{\mathscr{B}}$ and $s \in S^{\mathscr{C}}$, $\gamma^{\mathscr{C}}(s) = \gamma^{\mathscr{A}}(s_1) \wedge \gamma^{\mathscr{B}}(s_2)$,

(vi) $Inv^{\mathscr{C}} = Inv^{\mathscr{A}} \wedge Inv^{\mathscr{B}}$,

(vii) For all $s_1^{\mathscr{A}}$, $s_2^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_1^{\mathscr{B}}$, $s_2^{\mathscr{B}} \in S^{\mathscr{B}}$, $\phi^{\mathscr{C}} = \phi^{\mathscr{A}} \wedge \phi^{\mathscr{B}}$, and $Y^{\mathscr{C}} = Y^{\mathscr{A}} \cup Y^{\mathscr{B}}$: $((s_1^{\mathscr{A}}, s_2^{\mathscr{A}}), \phi^{\mathscr{C}}, Y^{\mathscr{C}}, (s_1^{\mathscr{B}}, s_2^{\mathscr{B}})) \in \rightarrow_{icta}^{\mathscr{C}}$ iff there exist transitions $(s_1^{\mathscr{A}}, \phi^{\mathscr{A}}, Y^{\mathscr{A}}, s_2^{\mathscr{A}}) \in \rightarrow_{icta}^{\mathscr{A}}$ and $(s_1^{\mathscr{B}}, \phi^{\mathscr{B}}, Y^{\mathscr{B}}, s_2^{\mathscr{B}}) \in \rightarrow_{icta}^{\mathscr{B}}$,

(viii) $F^{\mathscr{C}} = F^{\mathscr{A}} \times F^{\mathscr{B}}$,

(ix) $\pi^{\mathscr{C}} = \pi^{\mathscr{A}} \cup \pi^{\mathscr{B}}$,

The proof of intersection correctness is the following: We need to show that for any $\tau \in$ *Rates*, $\rho \in \mathscr{L}(\mathscr{C},\tau)$ iff $\rho \in \mathscr{L}(\mathscr{A},\tau) \cap \mathscr{L}(\mathscr{B},\tau)$. Assume that $\rho = (\gamma(\zeta_0), \{t_0\})$, $(\gamma(s_1),]t_1, t_2[)$, ..., $(\gamma(s_n),]t_{n-1}, t_n[) \in \mathscr{L}(\mathscr{C},\tau)$ is an TIS and an icTA $\mathscr{C}$. Then ρ originates from some runs $\theta = (s_0^{\mathscr{C}}, \nu_0, t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{C}}, \nu_1, t_1) \ldots \xrightarrow{\zeta_{n-1}} (s_n^{\mathscr{C}}, \nu_n, t_n)$ of $\mathscr{C}$ with regard to τ, where a run is an alternating sequence of states and transitions. The states are triples of a location, a clock valuation, and lastly the reference time: $\{(s,\nu,t) \in S^{\mathscr{C}} \times \mathbb{R}_{\geq 0}^{X} \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s)\}$. The transitions must alternate between two types: (i) Delay transition, i.e. spending time in a location: $q_i \xrightarrow{d} q_i + d$, where $q_i = (s_i, \nu_i, t_i)$, and $q_i + d = (s_i, \nu_i + (\tau(t_i + d) - \tau(t_i)), t_i + d)$, if the invariant is continuously true in local time: $\forall t \in]t_i, t_i + d[: \nu_i + (\tau(t) - \tau(t_i)) \models Inv(s_i)$. (ii) Discrete transition: following a transition $\zeta_i = (s_{i-1}, \phi, Y, s_i) \in \rightarrow_{\text{icTA}}$ when the clock constraint ϕ is satisfied. The clocks in Y are then reset. This transition is instantaneous. $(s_{i-1}, \nu_{i-1}, t_{i-1}) \xrightarrow{\zeta_i} (s_i, \nu_i, t_i)$, such that $\nu_{i-1} \models \phi$, $\nu_i = \nu_{i-1}[Y \rightarrow 0]$, $t_{i-1} = t_i$. For any clock $x^{\mathscr{C}} \in X^{\mathscr{C}}$, $\nu(x^{\mathscr{C}}) = \tau(t) - \tau(t_i)$ where $i \geq 0$ is the index of the last transition which reset clock $x^{\mathscr{C}}$ or is $\tau(t)$ if $x^{\mathscr{C}}$ was never reset. We can say the $\mathscr{C}$ accepts ρ at time t with regard to τ, if there is a run θ for an equivalent of ρ that visits an accepting location $s_n^{\mathscr{C}} \in F^{\mathscr{C}}$ at t and on the *rate* τ, where we can safely assume due to the construction that, for each $1 \leq i \leq n$, $s_0^{\mathscr{C}} \xrightarrow{\rho}_{\mathscr{C}} s_n^{\mathscr{C}}$. By induction hypothesis we know that $\rho = (\gamma(\zeta_0), \{t_0\})$, $(\gamma(s_1),]t_1, t_2[), \ldots, (\gamma(s_n),]t_{n-1}, t_n[) \in (\mathscr{L}(\mathscr{A},\tau) \cap \mathscr{L}(\mathscr{B},\tau))$, then there is an accepting run $\theta' = (s_0^{\mathscr{A}\cap\mathscr{B}}, \nu'_0, t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{A}\cap\mathscr{B}}, \nu'_1, t_1) \ldots \xrightarrow{\zeta_{n-1}} (s_n^{\mathscr{A}\cap\mathscr{B}}, \nu'_n, t_n)$ be the run for ρ that visits an accepting location $s_n^{\mathscr{A}\cap\mathscr{B}} \in (F^{\mathscr{A}} \times F^{\mathscr{B}})$ at time t and on the rate τ, where we can safely assume due to the construction that, for each

$1 \leq i \leq n$, $s_0^{\mathscr{A}\cap\mathscr{B}} \xrightarrow{\rho}_{\mathscr{A}\cap\mathscr{B}} s_n^{\mathscr{A}\cap\mathscr{B}}$. Firstly, we say that $\mathscr{C}$ accepts an TIS ρ at t with regard to *rate* τ, if there is a run θ for ρ that visits a accepting location $s_n \in F^{\mathscr{C}}$ at t. Secondly the clock valuation depends on the TIS ρ, on the reference time of evaluation t, and on the *rate* τ. It is easy to see that for each $1 \leq i \leq n$ the clock valuation assigns a (non-standard) positive real to each clock variable $v_i = v_i^{'}$. Thirdly, since $s_{n-1}^{\mathscr{C}} \in (S^{\mathscr{A}} \times S^{\mathscr{B}})$ and there exists a transition $\zeta_{n-1}^{\mathscr{C}} = (s_{n-2}, \phi, Y, s_{n-1}) \in \rightarrow_{icTA}^{\mathscr{C}}$, there exists $s_n^{\mathscr{A}\cap\mathscr{B}} \in (F^{\mathscr{A}} \times F^{\mathscr{B}})$ such that $s_n^{\mathscr{C}} \in (F^{\mathscr{A}} \times F^{\mathscr{B}})$ and there exists a transition $\zeta_n^{\mathscr{A}\cap\mathscr{B}} = ((s_{n-1}^{\mathscr{A}}, s_n^{\mathscr{A}}), \phi^{\mathscr{C}}, Y^{\mathscr{C}}, (s_{n-1}^{\mathscr{B}}, s_n^{\mathscr{B}})) \in\rightarrow_{icta}^{\mathscr{C}}$ iff there exist transitions $(s_{n-1}^{\mathscr{A}}, \phi^{\mathscr{A}}, Y^{\mathscr{A}}, s_n^{\mathscr{A}}) \in \rightarrow_{icta}^{\mathscr{A}}$ and $(s_{n-1}^{\mathscr{B}}, \phi^{\mathscr{B}}, Y^{\mathscr{B}}, s_n^{\mathscr{B}}) \in \rightarrow_{icta}^{\mathscr{B}}$, for all $s_{n-1}^{\mathscr{A}}$, $s_n^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_{n-1}^{\mathscr{B}}$, $s_n^{\mathscr{B}} \in S^{\mathscr{B}}$, $\phi^{\mathscr{C}} = \phi^{\mathscr{A}} \wedge \phi^{\mathscr{B}}$, and $Y^{\mathscr{C}} = Y^{\mathscr{A}} \cup Y^{\mathscr{B}}$, where the clock constraints are satisfied by the valuation $v_n^{'}$. Hence we can adjunct $s_0^{\mathscr{A}\cap\mathscr{B}} \xrightarrow{\rho} s_n^{\mathscr{A}\cap\mathscr{B}}$ to the run over $\mathscr{A}$ and $\mathscr{B}$. It follows immediately that $\mathscr{C}$ accepts the set of TIS ρ at time t and on the *rate* τ, then the language $\mathscr{L}(\mathscr{C}, \tau)$ is the set of TIS ρ of accepting runs of $\mathscr{C}$ with regard to τ, and the language $\mathscr{L}(\mathscr{A}, \tau) \cap \mathscr{L}(\mathscr{B}, \tau)$ is defined as the set of TIS ρ of accepting runs of the intersection of $\mathscr{A}$ and $\mathscr{B}$ with regard to τ. We have that $\mathscr{L}(\mathscr{C}, \tau) = \mathscr{L}(\mathscr{A}, \tau) \cap \mathscr{L}(\mathscr{B}, \tau)$. □

Theorem 52. *If $\mathscr{C}$ is a τ-union of $\mathscr{A}, \mathscr{B}$, then for any $P \subseteq Proc$, $\mathscr{L}_{\exists}(\mathscr{C}, P) = \mathscr{L}_{\exists}(\mathscr{A}, P) \cup \mathscr{L}_{\exists}(\mathscr{B}, P)$.*

Proof.

$$\begin{aligned}
\mathscr{L}_{\exists}(\mathscr{A}, P) \cup \mathscr{L}_{\exists}(\mathscr{B}, P) &= \Big(\bigcup_{\tau \in Rates} \mathscr{L}(\mathscr{A}, \tau, P)\Big) \cup \Big(\bigcup_{\tau \in Rates} \mathscr{L}(\mathscr{B}, \tau, P)\Big) \\
&= \Big(\bigcup_{\tau \in Rates} \tau_P(\mathscr{L}(\mathscr{A}, \tau))\Big) \cup \Big(\bigcup_{\tau \in Rates} \tau_P(\mathscr{L}(\mathscr{B}, \tau))\Big) \\
&= \Big(\bigcup_{\tau \in Rates} \{\tau_P(\rho) \mid \rho \in \mathscr{L}(\mathscr{A}, \tau)\}\Big) \cup \\
&\quad \Big(\bigcup_{\tau \in Rates} \{\tau_P(\rho) \mid \rho \in \mathscr{L}(\mathscr{B}, \tau)\}\Big) \\
&= \{(\sigma, \tau_P(I)) \mid (\rho \in \mathscr{L}(\mathscr{A}, \tau)) \vee (\rho \in \mathscr{L}(\mathscr{B}, \tau)) \\
&\quad \wedge\ \tau \in Rates\} \\
&= \{(\sigma, \tau_P(I)) \mid \rho \in (\mathscr{L}(\mathscr{A}, \tau) \cup \mathscr{L}(\mathscr{B}, \tau)) \\
&\quad \wedge\ \tau \in Rates\} \\
&= \{(\sigma, \tau_P(I)) \mid \rho \in \mathscr{L}(\mathscr{C}, \tau) \wedge\ \tau \in Rates\} \\
&= \mathscr{L}_{\exists}(\mathscr{C}, P)
\end{aligned}$$

□

Theorem 53. *If $\mathscr{C}$ is an τ-intersection of $\mathscr{A}, \mathscr{B}$, then for any $P \subseteq Proc$, $\mathscr{L}_{\forall}(\mathscr{C}, P) = \mathscr{L}_{\forall}(\mathscr{A}, P) \cap \mathscr{L}_{\forall}(\mathscr{B}, P)$.*

Proof.

$$\begin{aligned}
\mathscr{L}_{\forall}(\mathscr{A},P)\cap\mathscr{L}_{\forall}(\mathscr{B},P) &= (\bigcap_{\tau\in Rates}\mathscr{L}(\mathscr{A},\tau,P))\cap(\bigcap_{\tau\in Rates}\mathscr{L}(\mathscr{B},\tau,P))\\
&= (\bigcap_{\tau\in Rates}\tau_P(\mathscr{L}(\mathscr{A},\tau)))\cap(\bigcap_{\tau\in Rates}\tau_P(\mathscr{L}(\mathscr{B},\tau)))\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{A},\tau)\})\cap\\
&\quad (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{B},\tau)\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid(\rho\in\mathscr{L}(\mathscr{A},\tau))\wedge
\end{aligned}$$

$$\begin{aligned}
&\quad (\rho\in\mathscr{L}(\mathscr{B},\tau))\}\wedge\ \tau\in \mathit{Rates}\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in(\mathscr{L}(\mathscr{A},\tau)\cap\\
&\quad \mathscr{L}(\mathscr{B},\tau))\}\wedge\ \tau\in \mathit{Rates}\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{C},\tau)\wedge\ \tau\in \mathit{Rates}\})\\
&= \mathscr{L}_{\forall}(\mathscr{C},P)
\end{aligned}$$

□

Theorem 54. *If $\mathscr{C}$ is an τ-intersection of $\mathscr{A},\mathscr{B}$, then for any $P\subseteq Proc$, $\mathscr{L}_{\exists}(\mathscr{C},P)=\mathscr{L}_{\exists}(\mathscr{A},P)\cap\mathscr{L}_{\exists}(\mathscr{B},P)$.*

Proof.

$$\begin{aligned}
\mathscr{L}_{\exists}(\mathscr{A},P)\cap\mathscr{L}_{\forall}(\mathscr{B},P) &= (\bigcap_{\tau\in Rates}\mathscr{L}(\mathscr{A},\tau,P))\cap(\bigcap_{\tau\in Rates}\mathscr{L}(\mathscr{B},\tau,P))\\
&= (\bigcap_{\tau\in Rates}\tau_P(\mathscr{L}(\mathscr{A},\tau)))\cap(\bigcap_{\tau\in Rates}\tau_P(\mathscr{L}(\mathscr{B},\tau)))\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{A},\tau)\})\cap\\
&\quad (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{B},\tau)\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid(\rho\in\mathscr{L}(\mathscr{A},\tau))\wedge\\
&\quad (\rho\in\mathscr{L}(\mathscr{B},\tau))\}\wedge\ \tau\in \mathit{Rates}\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in(\mathscr{L}(\mathscr{A},\tau)\cap\\
&\quad \mathscr{L}(\mathscr{B},\tau))\}\wedge\ \tau\in \mathit{Rates}\})\\
&= (\bigcap_{\tau\in Rates}\{\tau_P(\rho)\mid\rho\in\mathscr{L}(\mathscr{C},\tau)\wedge\ \tau\in \mathit{Rates}\})\\
&= \mathscr{L}_{\exists}(\mathscr{C},P)
\end{aligned}$$

□

Theorem 55. *If $\mathscr{C}$ is an τ-union of $\mathscr{A}, \mathscr{B}$, then for any $P \subseteq Proc$, $\mathscr{L}_\forall(\mathscr{C}, P) = \mathscr{L}_\forall(\mathscr{A}, \tau) \cup \mathscr{L}_\forall(\mathscr{B}, P)$.*

Proof.

$$\begin{aligned}
\mathscr{L}_\forall(\mathscr{A}, P) \cup \mathscr{L}_\exists(\mathscr{B}, P) &= (\bigcup_{\tau \in Rates} \mathscr{L}(\mathscr{A}, \tau, P)) \cup (\bigcup_{\tau \in Rates} \mathscr{L}(\mathscr{B}, \tau, P)) \\
&= (\bigcup_{\tau \in Rates} \tau_P(\mathscr{L}(\mathscr{A}, \tau))) \cup (\bigcup_{\tau \in Rates} \tau_P(\mathscr{L}(\mathscr{B}, \tau)))
\end{aligned}$$

$$\begin{aligned}
&= (\bigcup_{\tau \in Rates} \{\tau_P(\rho) \mid \rho \in \mathscr{L}(\mathscr{A}, \tau)\}) \cup \\
&\quad (\bigcup_{\tau \in Rates} \{\tau_P(\rho) \mid \rho \in \mathscr{L}(\mathscr{B}, \tau)\}) \\
&= \{(\sigma, \tau_P(I)) \mid (\rho \in \mathscr{L}(\mathscr{A}, \tau)) \vee (\rho \in \mathscr{L}(\mathscr{B}, \tau)) \\
&\quad \wedge \tau \in Rates\} \\
&= \{(\sigma, \tau_P(I)) \mid \rho \in (\mathscr{L}(\mathscr{A}, \tau) \cup \mathscr{L}(\mathscr{B}, \tau)) \\
&\quad \wedge \tau \in Rates\} \\
&= \{(\sigma, \tau_P(I)) \mid \rho \in \mathscr{L}(\mathscr{C}, \tau) \wedge \tau \in Rates\} \\
&= \mathscr{L}_\forall(\mathscr{C}, P)
\end{aligned}$$

□

We note that the above theorems are valid for the finite automata, but they are not correct for the ω-automata [Pit06]. The closure under the union operation for the ω-automata icTA can be done similar to 50. The intersection for the ω-automata icTA can be done using the same construction of intersection for the Büchi automata. We denote the ω-automata icTA as icTA$_I$.

Theorem 56. *icTA$_I$ is closed under intersection operation.*

Proof. Let $\mathscr{A} = (S^{\mathscr{A}}, s_0^{\mathscr{A}}, \Sigma^{\mathscr{A}}, X^{\mathscr{A}}, \gamma^{\mathscr{A}}, Inv^{\mathscr{A}}, \rightarrow_{icta}^{\mathscr{A}}, F^{\mathscr{A}}, \pi^{\mathscr{A}})$ and $\mathscr{B} = (S^{\mathscr{B}}, s_0^{\mathscr{B}}, \Sigma^{\mathscr{B}}, X^{\mathscr{B}}, \gamma^{\mathscr{B}}, Inv^{\mathscr{B}}, \rightarrow_{icta}^{\mathscr{B}}, F^{\mathscr{B}}, \pi^{\mathscr{B}})$ be Büchi icTA. Without loss of generality we assume that the sets of clocks $X^{\mathscr{A}}$ and $X^{\mathscr{B}}$ (and respectively the sets of locations $S^{\mathscr{A}}$ and $S^{\mathscr{B}}$) are all pairwise disjoint. Let $\mathscr{C} = (\Sigma^{\mathscr{C}}, X^{\mathscr{C}}, S^{\mathscr{C}}, s_0^{\mathscr{C}}, \rightarrow_{icta}^{\mathscr{C}}, \gamma^{\mathscr{C}}, Inv^{\mathscr{C}}, F^{\mathscr{C}}, \pi^{\mathscr{C}})$ be the icTA defined as follows:

(i) $S^{\mathscr{C}} = S^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1,2\}$,

(ii) $s_0^{\mathscr{C}} = (s_0^{\mathscr{A}}, s_0^{\mathscr{B}}, 1)$,

(iii) $\Sigma^{\mathscr{C}} = \Sigma^{\mathscr{A}} \cap \Sigma^{\mathscr{B}}$,

(iv) $X^{\mathscr{C}} = X^{\mathscr{A}} \cup X^{\mathscr{B}}$,

(v) For all $s_1^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_2^{\mathscr{B}} \in S^{\mathscr{B}}$ and $s \in S^{\mathscr{C}}$, $\gamma^{\mathscr{C}}(s) = \gamma^{\mathscr{A}}(s_1) \wedge \gamma^{\mathscr{B}}(s_2)$,

(vi) $Inv^{\mathscr{C}} = Inv^{\mathscr{A}} \wedge Inv^{\mathscr{B}}$,

(vii) For all $s_1^{\mathscr{A}}$, $s_2^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_1^{\mathscr{B}}$, $s_2^{\mathscr{B}} \in S^{\mathscr{B}}$, $\phi^{\mathscr{C}} = \phi^{\mathscr{A}} \wedge \phi^{\mathscr{B}}$, and $Y^{\mathscr{C}} = Y^{\mathscr{A}} \cup Y^{\mathscr{B}}$ and $i, j \in \{1,2\}$: $((s_1^{\mathscr{A}}, s_1^{\mathscr{B}}, i), \phi^{\mathscr{C}}, Y^{\mathscr{C}}, (s_2^{\mathscr{A}}, s_2^{\mathscr{B}}, j)) \in \rightarrow_{icta}^{\mathscr{C}}$ iff there exist transitions $(s_1^{\mathscr{A}}, \phi^{\mathscr{A}}, Y^{\mathscr{A}}, s_2^{\mathscr{A}}) \in \rightarrow_{icta}^{\mathscr{A}}$ and $(s_1^{\mathscr{B}}, \phi^{\mathscr{B}}, Y^{\mathscr{B}}, s_2^{\mathscr{B}}) \in \rightarrow_{icta}^{\mathscr{B}}$ and:

(a) if $i = 1$ and $s_1^{\mathscr{A}} \in F^{\mathscr{A}}$, then $j = 2$, or

(b) if $i = 2$ and $s_1^{\mathscr{B}} \in F^{\mathscr{B}}$, then $j = 1$, or

(c) neither *a*) or *b*) above applies and $j = i$.

(viii) $F^{\mathscr{C}} = F^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1\}$,

(ix) $\pi^{\mathscr{C}} = \pi^{\mathscr{A}} \cup \pi^{\mathscr{B}}$,

The proof of intersection correctness to infinite timed sequence is the following: We need to show that for any $\tau \in$ *Rates*, $\rho \in \mathscr{L}(\mathscr{C},\tau)$ iff $\rho \in \mathscr{L}(\mathscr{A},\tau) \cap \mathscr{L}(\mathscr{B},\tau)$. Assume that $\rho = (\gamma(\zeta_0), \{t_0\}), (\gamma(s_1),]t_1, t_2[), \ldots \in \mathscr{L}(\mathscr{C},\tau)$ is an TIS and an icTA $\mathscr{C}$. Then ρ originates from some runs $\theta = (s_0^{\mathscr{C}}, v_0, t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{C}}, v_1, t_1) \ldots$ of $\mathscr{C}$ with regards to τ, where a run is an alternating sequence of states and transitions. The states are triples of a location, a clock valuation, and the reference time: $\{(s, v, t) \in S^{\mathscr{C}} \times \mathbb{R}_{\geq 0}^X \times \mathbb{R}_{\geq 0} \mid v \models Inv(s)\}$. The transitions must alternate between two types: (*i*) Delay transition, i.e. spending time in a location: $q_i \xrightarrow{d} q_i + d$, where $q_i = (s_i, v_i, t_i)$, and $q_i + d = (s_i, v_i + (\tau(t_i + d) - \tau(t_i)), t_i + d)$, if the invariant is continuously true in local time: $\forall t \in]t_i, t_i + d[: v_i + (\tau(t) - \tau(t_i)) \models Inv(s_i)$. (*ii*) Discrete transition: following a transition $\zeta_i = (s_{i-1}, \phi, Y, s_i) \in \rightarrow_{\mathsf{icTA}}$ when the clock constraint ϕ is satisfied. The clocks in Y are then reset. This transition is instantaneous. $(s_{i-1}, v_{i-1}, t_{i-1}) \xrightarrow{\zeta_i} (s_i, v_i, t_i)$, such that $v_{i-1} \models \phi$, $v_i = v_{i-1}[Y \rightarrow 0], t_{i-1} = t_i$. For any clock $x^{\mathscr{C}} \in X^{\mathscr{C}}$, $v(x^{\mathscr{C}}) = \tau(t) - \tau(t_i)$ where $i \geq 0$ is the index of the last transition which reset clock $x^{\mathscr{C}}$ or is $\tau(t)$ if $x^{\mathscr{C}}$ was never reset. To determine whether a run of ρ is accepting, we consider the set $\mathit{inf}(\rho) \subseteq F^{\mathscr{C}}$ which is the set of all locations that occur in ρ infinitely often. We can say $\mathscr{C}$ accepts ρ at time t with regards to τ, if there is a run θ for an equivalent of ρ that visits infinitely often accepting locations in $F^{\mathscr{C}}$ at t and on the *rate* τ, where we can safely assume due to the construction that, $\mathit{inf}(\rho) \cap F^{\mathscr{C}} \neq \emptyset$. By induction hypothesis we know that $\rho = (\gamma(\zeta_0), \{t_0\}), (\gamma(s_1),]t_1, t_2[), \ldots \in (\mathscr{L}(\mathscr{A},\tau) \cap \mathscr{L}(\mathscr{B},\tau))$, then there is an accepting run $\theta' = (s_0^{\mathscr{A}\cap\mathscr{B}}, v_0', t_0) \xrightarrow{\zeta_0} (s_1^{\mathscr{A}\cap\mathscr{B}}, v_1', t_1) \ldots$ be the run for ρ at time t and on the rate τ, where we can safely assume due to the accepting condition of Büchi automaton that, $\mathscr{A}$ universally accepts ρ if, $\mathit{inf}(\rho) \cap F^{\mathscr{A}\cap\mathscr{B}} \neq \emptyset$, where $\mathit{inf}(\rho)$ is set of locations that occur infinitely often in ρ. Firstly, we say that $\mathscr{C}$ accepts an TIS ρ at t with regards to *rate* τ, if there is a run θ for ρ that visits infinitely often accepting locations in $F^{\mathscr{A}\cap\mathscr{B}}$ at t. Secondly the clock valuation depends on the TIS ρ, on the reference time of evaluation t, and on the *rate* τ. It is easy to see that for each $i \geq 0$ the clock valuation assigns a (non-standard) positive real to each clock variable $v_i = v_i'$. Thirdly, since $s^{\mathscr{C}} \in (S^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1, 2\})$ and there exists a transition $\zeta^{\mathscr{C}} = (s, \phi, Y, s') \in \rightarrow_{icta}^{\mathscr{C}}$, there exists a $s^{\mathscr{A}\cap\mathscr{B}}$ that visit infinitely often to $(F^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1\})$ such that $s^{\mathscr{C}} \in (S^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1,2\})$ and there exists a transition $\zeta^{\mathscr{A}\cap\mathscr{B}} = ((s_1^{\mathscr{A}}, s_1^{\mathscr{B}}, i), \phi^{\mathscr{C}}, Y^{\mathscr{C}}, (s_2^{\mathscr{A}}, s_2^{\mathscr{B}}, j)) \in \rightarrow_{icta}^{\mathscr{C}}$ iff there exist transitions $(s_1^{\mathscr{A}}, \phi^{\mathscr{A}}, Y^{\mathscr{A}}, s_2^{\mathscr{A}}) \in \rightarrow_{icta}^{\mathscr{A}}$ and $(s_1^{\mathscr{B}}, \phi^{\mathscr{B}}, Y^{\mathscr{B}}, s_2^{\mathscr{B}}) \in \rightarrow_{icta}^{\mathscr{B}}$ for all $s_1^{\mathscr{A}}$, $s_2^{\mathscr{A}} \in S^{\mathscr{A}}$, $s_1^{\mathscr{B}}$, $s_2^{\mathscr{B}} \in S^{\mathscr{B}}$, $\phi^{\mathscr{C}} = \phi^{\mathscr{A}} \wedge \phi^{\mathscr{B}}$, and $Y^{\mathscr{C}} = Y^{\mathscr{A}} \cup Y^{\mathscr{B}}$ and i, $j \in \{1,2\}$, and:

(i) if $i = 1$ and $s_1^{\mathscr{A}} \in F^{\mathscr{A}}$, then $j = 2$, or

(ii) if $i = 2$ and $s_1^{\mathscr{B}} \in F^{\mathscr{B}}$, then $j = 1$, or

(iii) Neither *a*) or *b*) above applies and $j = i$.

Hence we can adjunct $\mathit{inf}(\rho) \cap (F^{\mathscr{A}} \times S^{\mathscr{B}} \times \{1\})$ to the run over $\mathscr{A}$ and $\mathscr{B}$. It follows immediately that $\mathscr{C}$ accepts the set of TIS ρ at time t and on the *rate* τ, then the language $\mathscr{L}(\mathscr{C}, \tau)$ is the set of TIS ρ of accepting runs of $\mathscr{C}$ with regards to τ, and the language $\mathscr{L}(\mathscr{A}, \tau) \cap \mathscr{L}(\mathscr{B}, \tau)$ is defined as the set of TIS ρ of accepting runs of the intersection of $\mathscr{A}$ and $\mathscr{B}$ with regards to τ. We have that $\mathscr{L}(\mathscr{C}, \tau) = \mathscr{L}(\mathscr{A}, \tau) \cap \mathscr{L}(\mathscr{B}, \tau)$. □

However, icTA$_I$ are not determinizable, not closed under complement, and their inclusion problem is undecidable (whether τ-wise, existential with at least one observer, or universal with at most one observer), because TA [AD94] are a special case of icTA.

6.3 Recursive Distributed Event Clocks Temporal Logic

This section aims to construct a fully decidable distributed real-time logic to specify requirements on DRTS. (Recursive) Distributed Event Clock Temporal Logic (DECTL) extend the (Recursive) Event Clock Temporal Logic (EventClockTL) [RS97, HRS98] with distributed (a.k.a. independent) clocks. As in Section 6.1, we assume a set of processes *Proc*. The clocks of each process will evolve according to its local time given by a rate τ. DECTL is based on LTL, and adds two local real-time modalities. The recording modality $\triangleleft_I^q \phi$ means that ϕ was true last time in the interval I according to the local time of q. Symmetrically, the predicting modality $\triangleright_I^q \phi$ says the ϕ will occur within I according to the local time of q. If we have only one process, we find back EventClockTL [RS97].

We could construct similarly a more expressive logic that allows us to observe not only the last ϕ, but also the last but one, and more generally the last but n ϕ [OLS10]. This logic is still translatable in DECA. We could also extend the known expressive equivalence of EventClockTL and MITL+Past [HRS98] to construct a version of MITL with independent clocks. Lastly, independent clocks can also be introduced in a linear μ-calculus, to increase expressiveness by counting.

Definition 68. *The formulas of DECTL are defined by the grammar:*

$$\phi ::= \ true \mid p \in \mathbb{P} \mid \triangleright_I^q \ \phi \mid \ \triangleleft_I^q \ \phi \mid \phi_1 \ \wedge \ \phi_2 \mid \neg\phi \mid \phi_1 \ \mathscr{U} \ \phi_2 \mid \phi_1 \ \mathscr{S} \ \phi_2$$

where p is a propositional symbol, $I \in \mathscr{I}_{\mathbb{N}}$ is an interval and $q \in Proc$. We can now define how to evaluate the truth value of a DECTL formula along an TIS ρ and a rate τ, noted $(\rho, t) \models_\tau \phi_1$. We omit τ below.

$$\begin{array}{lcl}
(\rho,t) \models & p & \textit{iff}\ p \in \rho(t) \\
(\rho,t) \models & \neg\phi & \textit{iff}\ (\rho,t) \not\models \phi \\
(\rho,t) \models & \phi_1 \wedge \phi_2 & \textit{iff}\ (\rho,t) \models \phi_1 \ \textit{and}\ (\rho,t) \models \phi_2 \\
(\rho,t) \models & \phi_1 \mathscr{U} \phi_2 & \textit{iff}\ \exists t' > t.(\rho,t') \models \phi_2 \ \textit{and}\ \forall t'' \in (t,t'),(\rho,t'') \models \phi_1 \\
(\rho,t) \models & \phi_1 \mathscr{S} \phi_2 & \textit{iff}\ \exists t' < t.(\rho,t') \models \phi_2 \ \textit{and}\ \forall t'' \in (t',t),(\rho,t'') \models \phi_1 \\
(\rho,t) \models & \triangleleft_I^q \phi & \textit{iff}\ \exists t' < t.\tau_q(t) - \tau_q(t') \in I \wedge (\rho,t') \models \phi \ \textit{and}\ \forall t'' < t.\tau_q(t) - \\
 & & \tau_q(t'') < I,(\rho,t'') \not\models \phi \\
(\rho,t) \models & \triangleright_I^q \phi & \textit{iff}\ \exists t' > t.\tau_q(t') - \tau_q(t) \in I \wedge (\rho,t') \models \phi \ \textit{and}\ \forall t'' > t.\tau_q(t'') - \\
 & & \tau_q(t) < I,(\rho,t'') \not\models \phi
\end{array}$$

Example 19. *The formula $\neg(\mathscr{F} b \wedge \neg \rhd^{q}_{\leq 1} b)$, where $\mathscr{F} b = true\, \mathscr{U} b$ says that the first b, if any, must occur within 1 second, as measured by q. It holds on the automation of Figure 6.1. However, the formula measured by p, $\neg(\mathscr{F} b \wedge \neg \rhd^{p}_{\leq 1} b)$, does not hold.*

6.3.1 From DECTL and DECA

In this section, we show that DECA are sufficiently expressive to define all DECTL properties. We can translate any DECTL formula ϕ into a DECA automaton $\mathscr{A}_\phi$ that accepts the pairs (ρ, t), such that $(\rho, t) \models_\tau \phi$, for all τ by a tableau construction. The translation is done level by level, where the level of a formula is the nesting depth of real-time modalities. A formula $\rhd^{q}_{I}\phi$ is translated as constraint $x^{q}_{\mathscr{A}_\phi} \in I$. The formula ϕ is recursively translated in a tableau automaton $\mathscr{A}_\phi$ where the monitored states are the states containing ϕ.

Definition 69. *The level of a DECTL formulas ϕ_1, ϕ_2, ϕ_3 denoted by $\mathscr{LF}$, is a recursive function that satisfy the following:*

$$\begin{array}{lcl}
\mathscr{LF}(p) & = & 0 \\
\mathscr{LF}(\phi_1 \vee \phi_2) & = & Max(\mathscr{LF}(\phi_1) \cup \mathscr{LF}(\phi_2)) \\
\mathscr{LF}(\neg\phi_1) & = & \mathscr{LF}(\phi_1) \\
\mathscr{LF}(\phi_1 \mathscr{U} \phi_2) & = & Max(\mathscr{LF}(\phi_1) \cup \mathscr{LF}(\phi_2))) \\
\mathscr{LF}(\phi_1 \mathscr{S} \phi_2) & = & Max(\mathscr{LF}(\phi_1) \cup \mathscr{LF}(\phi_2)) \\
\mathscr{LF}(\rhd^{q}_{I}\phi_3) & = & 1 + \mathscr{LF}(\phi_3) \\
\mathscr{LF}(\lhd^{q}_{I}\phi_3) & = & 1 + \mathscr{LF}(\phi_3)
\end{array}$$

A formula ϕ is of level i, if $\mathscr{LF}(\phi) = i$. Recursively, we can define that the formula ϕ_3 is a level k where $0 \leq k < i$ and ϕ_1, ϕ_2 is a level j where $0 \leq j \leq i$.

Definition 70. *The closure set of a DECTL formula ϕ_1, ϕ_2, ϕ_3 denoted by $\mathscr{CF}$, is a recursive function that satisfy the following:*

$$\begin{array}{lcl}
\mathscr{CF}(p) & = & \{p\} \\
\mathscr{CF}(\phi_1 \vee \phi_2) & = & \mathscr{CF}(\phi_1) \cup \mathscr{CF}(\phi_2) \cup \{\phi_1 \vee \phi_2\} \\
\mathscr{CF}(\neg\phi_1) & = & \mathscr{CF}(\phi_1) \\
\mathscr{CF}(\phi_1 \mathscr{U} \phi_2) & = & \mathscr{CF}(\phi_1) \cup \mathscr{CF}(\phi_2) \cup \{\phi_1 \mathscr{U} \phi_2\} \\
\mathscr{CF}(\phi_1 \mathscr{S} \phi_2) & = & \mathscr{CF}(\phi_1) \cup \mathscr{CF}(\phi_2) \cup \{\phi_1 \mathscr{S} \phi_2\} \\
\mathscr{CF}(\rhd^{q}_{I}\phi_3) & = & \bigcup_{q \in Proc} \rhd^{q}_{I}\phi_3 \\
\mathscr{CF}(\lhd^{q}_{I}\phi_3) & = & \bigcup_{q \in Proc} \lhd^{q}_{I}\phi_3 \\
\mathscr{CF}^{c}(\phi_1) & \textit{is the set} & \mathscr{CF}(\phi_1)\ \textit{closed by negation.}
\end{array}$$

We give a proof for the construction of a DECA $\mathscr{A}_\phi$, for every DECTL formula ϕ. This proof is an adaptation of the construction for EventClockTL and RECA.

Theorem 57. *For every DECTL formula ϕ, we can construct a DECA $\mathscr{A}_\phi$, that accepts the pairs (ρ, t), where ρ is defined on the set of propositions appearing in ϕ and a time $t \in \mathbb{R}_{\geq 0}$, such that $(\rho, t) \models_\tau \phi$.*

Proof. Let $\mathscr{L}\mathscr{F}(\phi) = 0$. We define a transition structure $\mathscr{B} = (Q, q_0, \rightarrow_{ts}, Q_F)$ that checks the semantics of the operators and propositions of level 0 formula. For the other levels i, such that $i > 0$, we will transform the transition structure into a DECA. Let DECTL formula ϕ, ϕ_1, ϕ_2, we define $\mathscr{B}$ as follows:

(i) The state S is the set of pairs (s, φ), where $s \in 2^{\mathscr{C}\mathscr{F}^c(\phi)}$ with $\top \in s$ and $\varphi \in \{open, sing\}$(indicating if the control can stay in the state for an open interval of time or just a singular interval of time) and the following properties are verified:

  (a) For all $\phi_1 \in \mathscr{C}\mathscr{F}^c(\phi)$: $\phi_1 \in s$ iff $\neg\, \phi_1 \notin s$.

  (b) For all $(\phi_1 \vee \phi_2) \in \mathscr{C}\mathscr{F}^c(\phi)$: $\phi_1 \vee \phi_2 \in s$ iff $\phi_1 \in s$ or $\phi_2 \in s$.

  (c) For all $(\phi_1\ \mathscr{U}\ \phi_2) \in \mathscr{C}\mathscr{F}^c(\phi)$:

    i. If $\phi_2 \in s$ and $\varphi = open$ then $\phi_1\ \mathscr{U}\ \phi_2 \in s$.

    ii. If $\phi_1\ \mathscr{U}\ \phi_2 \in s$ and $\varphi = open$ then $\phi_1 \in s$ or $\phi_2 \in s$.

  (d) For all $(\phi_1\ \mathscr{S}\ \phi_2) \in \mathscr{C}\mathscr{F}^c(\phi)$:

    i. If $\phi_2 \in s$ and $\varphi = open$ then $\phi_1\ \mathscr{S}\ \phi_2 \in s$.

    ii. If $\phi_1\ \mathscr{S}\ \phi_2 \in s$ and $\varphi = open$ then $\phi_1 \in s$ or $\phi_2 \in s$.

(ii) The initial state is the subset of pairs $(s, \varphi) \in Q$, such that $\varphi = sing$ and does not exists $\phi_1\ \mathscr{S}\ \phi_2 \in \mathscr{C}\mathscr{F}^c(\phi)$ and $\phi_1\ \mathscr{S}\ \phi_2 \in s$. That initial state is singular and it does not contains a since formula in positive form.

(iii) The transition relation $\rightarrow_{ts}$ is a subset $[(s_1, \varphi_1), (s_2, \varphi_2)]$ of $Q \times Q$ that respects the following restrictions:

  (a) $\varphi_1 = open$ and $\varphi_2 = sing$ or $\varphi_1 = sing$ and $\varphi_2 = open$.

  (b) The following rules express how until formulas are transferred form one state to the next of the transition structure:

    i. $\phi_1\ \mathscr{U}\ \phi_2 \in s_1 \wedge \varphi_1 = sing$ iff $\phi_1\ \mathscr{U}\ \phi_2 \in s_2$.

    ii. $\phi_1\ \mathscr{U}\ \phi_2 \in s_1 \wedge \varphi_1 = open \wedge \phi_2 \notin s_1$, implies $(\phi_1\ \mathscr{U}\ \phi_2 \in s_2 \wedge \phi_1 \in s_2) \vee \phi_2 \in s_2$.

    iii. $\phi_1 \in s_1 \wedge \varphi_1 = open \wedge (\phi_1 \in s_2 \vee (\phi_2 \in s_2 \wedge \phi_1\ \mathscr{U}\ \phi_2 \in s_2))$ implies $\phi_1\ \mathscr{U}\ \phi_2 \in s_1$.

  (c) The following are for the since formulas:

    i. $\phi_1\ \mathscr{S}\ \phi_2 \in s_2 \wedge \varphi_2 = sing$ iff $\phi_1\ \mathscr{S}\ \phi_2 \in s_1$.

    ii. $\phi_1\ \mathscr{S}\ \phi_2 \in s_2 \wedge \varphi_2 = open \wedge \phi_2 \notin s_2$ implies $\phi_2 \in s_1 \vee (\phi_1 \in s_1 \wedge (\phi_1\ \mathscr{S}\ \phi_2) \in s_1)$.

    iii. $\phi_1 \in s_2 \wedge \varphi_2 = open \wedge (\phi_2 \in s_1 \vee (\phi_1\ \mathscr{S}\ \phi_2) \in s_1)$ implies $\phi_1\ \mathscr{S}\ \phi_2 \in s_2$.

(iv) We use a generalized Büchi acceptance condition. For each formula $\phi_1\ \mathscr{U}\ \phi_2 \in \mathscr{C}\mathscr{F}^c(\phi)$, there is a set $Q_{F_{\phi_1 \mathscr{U} \phi_2}} = \{(s, \varphi) \mid \phi_1 \mathscr{U} \phi_2 \notin s \vee \phi_2 \in s\}$.

Now, we will transform the transition structure $\mathscr{B}$ into a DECA $\mathscr{A}_\phi$. We construct $\mathscr{A}_\phi = (S^{\mathscr{A}_\phi}, s_0^{\mathscr{A}_\phi}, \Sigma^{\mathscr{A}_\phi}, X^{\mathscr{A}_\phi}, M^{\mathscr{A}_\phi}, \gamma^{\mathscr{A}_\phi}, Inv^{\mathscr{A}_\phi}, \rightarrow_{deca}^{\mathscr{A}_\phi}, F^{\mathscr{A}_\phi}, \pi^{\mathscr{A}_\phi})$ as follows:

(i) The set of locations $S^{\mathscr{A}_\phi}$ is the set of pairs $((s, \varphi), \varsigma)$ such that:

  (a) $(s, \varphi) \in S^{\mathscr{A}_\phi}$.

  (b) ς is a label that is open iff $\varphi = open$.

  (c) The labeling is propositionally consistent with the formula in s, for all proposition $p \in \mathbb{P}$: $p \in \varsigma$ iff $p \in s$,

(ii) The initial location $s_0^{\mathscr{A}_\phi}$ is the subset of locations $((s,\varphi),\varsigma) \in S^{\mathscr{A}_\phi}$ such that $(s,\varphi) = s_0^{\mathscr{A}_\phi}$,

(iii) The set of symbols used by $\mathscr{A}_\phi$ is the set of propositional symbols that appear in the formula ϕ, $\Sigma^{\mathscr{A}_\phi} = \{p \mid p \in \mathscr{C}\mathscr{F}^c(\phi)\}$,

(iv) The set of clocks used by $\mathscr{A}_\phi$ is the set of clocks that appear in the formula ϕ, $C^{\mathscr{A}_\phi} = \{\triangleright_I^q \cup \triangleleft_I^q \mid q \in Proc\; and\; I\; is\; an\; interval\}$,

(v) The set $M^{\mathscr{A}_\phi}$ of monitored locations is the subset of locations $((s,\varphi),\varsigma) \in S^{\mathscr{A}_\phi}$ such that $\phi \in s$, that is the subset of locations where the formula ϕ is true,

(vi) The labeling function $\gamma^{\mathscr{A}_\phi}$ is defined as follows: $\gamma^{\mathscr{A}_\phi}(((s,\varphi),\varsigma)) = \varsigma$,

(vii) The clock constraints $Inv^{\mathscr{A}_\phi}$ is defined as follows: $Inv^{\mathscr{A}_\phi}(((s,\varphi),\varsigma)) = \varphi$, item The transition relation is the set of pairs $[((s_1,\varphi_1),\varsigma_1),((s_2,\varphi_2),\varsigma_2)]$ with $((s_i,\varphi_i),\varsigma_i) \in S^{\mathscr{A}_\phi}$ for $i \in \{1,2\}$, such that: $((s_1,\varphi_1),(s_2,\varphi_2)) \in \rightarrow_{deca}$,

(viii) We transfer in $\mathscr{A}_\phi$ the generalized Büchi acceptance condition of the transition structure $\mathscr{B}$: $F^{\mathscr{A}_\phi}$ is the set of sets of accepting locations $\{F_1, F_2, \cdots F_n\}$ where each F_i corresponds to a set of accepting states in S as follows: $F_i = \{((s,\varphi),\varsigma) \mid (s,\varphi) \in Q_F\}$.

□

The construction is exponential in the size of the non-real time part of the formula, but linear in the real-time part. The test of emptiness is done by the region construction presented in Section 6.1.6, that is exponential in the real-time part but linear for the rest.

proposition 8. *The satisfiability and validity problems for DECTL are decidable.*

Proof. The satisfiability of a DECTL formula ϕ can be decided by constructing $\mathscr{A}_\phi$, the automata for ϕ and testing if $\mathscr{L}(\mathscr{A}_\phi) \neq \emptyset$. Similarly the validity of a DECTL formula ϕ can be decided by constructing $\mathscr{A}_{\neg\phi}$, the automaton for the negation of ϕ and testing if $\mathscr{L}(\mathscr{A}_{\neg\phi}) = \emptyset$. □

proposition 9. *The automaton $\mathscr{A}_\phi$ has $n \cdot 2^{O(n \cdot log\ c \cdot n)}$ locations, where n is the length of the formula ϕ (the number of propositions, modal operators and logical connectives) and c is the largest constant appearing in the constraints in $\mathscr{A}_\phi$.*

Proof. The time complexity for checking the satisfiability and validity of ϕ is singly exponential in the length c of the maximal constant that appear in ϕ, and singly exponential in the number of real-time operator in ϕ. In Particular checking the satisfiability and validity of the DECTL formula ϕ are $2^{O(n \cdot log\ c \cdot n)}$, where n is the length of the formula ϕ (the number of propositions, real-time operators and logical connectives) and c is the largest constant appearing in ϕ. □

proposition 10. *Satisfiability and validity of DECTL are PSPACE-complete.*

Proof. The number of subformulas in ϕ is bounded by $|\phi|$; the size of the closure $\mathscr{C}\mathscr{F}(\phi)$ is $O(n)$. Consequently, the number of locations of the automaton $\mathscr{A}_\phi$ is $O(2^n)$ and the number of clocks is $O(n)$. The size of the maximal constant that appears in the clock constraints of $\mathscr{A}_\phi$ is c. From the region automaton $\mathsf{RG}(\mathscr{A}_\phi)$ which is a

Büchi automaton, it follows that the emptiness problem of $\mathscr{L}(\mathscr{A}_\phi)$ can be checked in time $O(2^n \cdot n! \cdot c^n)$. The PSPACE upper bound follows by the observation that the search in the region automaton can be performed without explicitly constructing the automaton $\mathscr{A}_\phi$. The PSPACE-hardness of DECTL follows from the PSPACE-hardness of propositional temporal logic with until [SC85]. □

proposition 11. *The model-checking problem for DECTL is PSPACE-complete.*

Proof. To prove PSPACE-hardness, we observe that, as with RECA, the satisfiability problem for DECTL can be reduced to the model-checking problem: the DECTL formula ϕ is unsatisfiable iff $\mathscr{L}(\mathscr{A}_U) \subseteq \mathscr{L}(\neg\emptyset)$ for the universal RECA $\mathscr{A}_U$, which accepts all possible TIS. □

6.4 Application of DECA and DECTL

In this section, we focus on some scenarios of distributed real-time systems for DECA. We introduce a simple communication protocol (Fault-Tolerant Protocol) and we show how this protocol can be modeled as a DECA. We also describe the properties of this protocol in DECTL.

6.4.1 Fault-Tolerant Protocol

Let us consider a DRTS consisting of application tasks running under an operating system while using multiple processors interconnected over the Internet. The crucial problem is to verify both the temporal properties (e.g., end-to-end response time) and the logical properties (e.g., unsafe state avoidance) of the applications involving two types of shared resources, the processor and the bus. The disadvantage of the traditional models (TA, ECA, RECA, EventClockTL) for specifying and verifying these systems is that they do not take into account the independent clocks of the tasks (e.g., the clocks of a task evolve synchronously but independently of the clocks of the other tasks). Figure 6.2 shows the communication protocol. The protocol is a simple system consisting of two processes connected via the Internet. There are two sender and two reader tasks running on each process. The clocks on each process are periodically invoked when a message needs to be sent. The clock activates the SenderTask, which sends a message to the Internet. Receipt of a message by a process causes activation of the SenderTask task.

6.4.2 Modeling the Fault-Tolerant Protocol in DECA

Figure 6.3 shows the communication protocol modeled as a DECA. The high-level automaton has the event clock variables and the lower-level automaton has the events. We will refer to the processes 1 and 2 of Figure 6.2 as p and q ($Proc = \{p, q\}$, and the set of propositions $\mathbb{P} = \{send, retry, ack\}$), and also to the clocks in processes p and q, which run at different speeds. The clocks are reset by the first monitored transition of $\mathscr{B}$. At position q_0 (lower lever automaton) the process waits for the event send. When the process receives this event, its control evolves to the

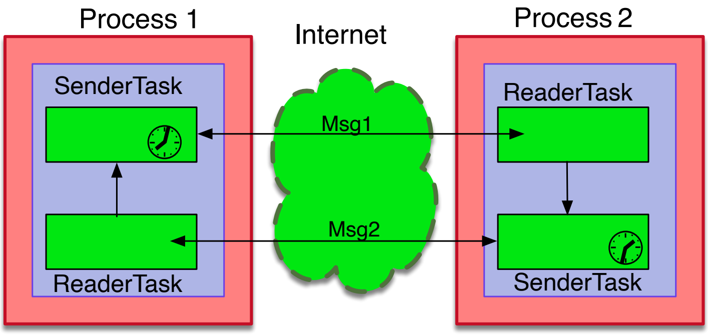


Figure 6.2: Fault-Tolerant Protocol

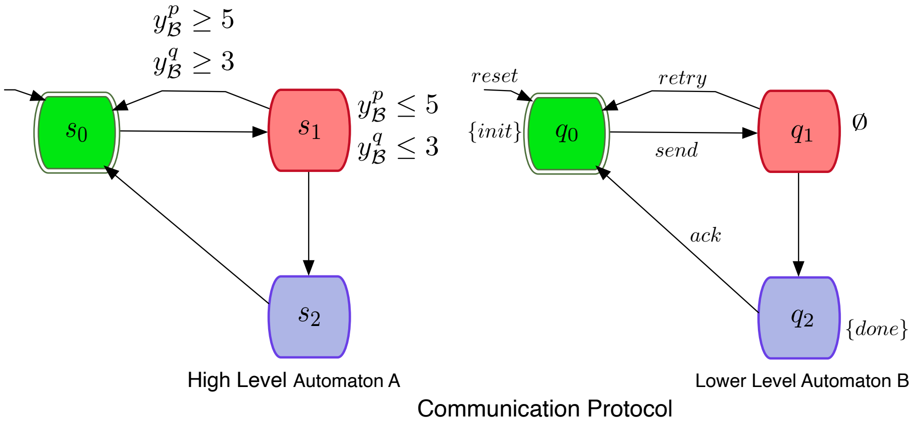


Figure 6.3: DECA Model of the Protocol

monitored location q_1 and the clock constraint $y^p_{\mathscr{B}} \leq 5$ of the high-level automaton imposes that the message must be sent before 5 time units for the process p or the clock constraint $y^q_{\mathscr{B}}. \leq 3$ of the high-level automaton imposes that the message must be sent before 3 time units for the process q. So this requirement imposes that the message takes less than 5 time units to go done for the process p and the message takes less than 3 time units to go done for the process q when it receives the information that a message is being sent. In q_1, the control must wait at least 5 time units before crossing the edge to location q_2, or the control must wait at least 3 time units before crossing the edge to location q_2. At q_2, the control must wait for the ack signal before crossing the edge to location q_3.

6.4.3 Properties of the Fault-Tolerant Protocol in DECTL

The property that a message is followed by an ack within 5 time units for the process p can be described by DECTL using the formula:

$$\Box(init \rightarrow \rhd^{p}_{\leq 5} done)$$

The property that a message is followed by an ack within 3 time units for the process q can be described by DECTL by the formula:

$$\Box(init \rightarrow \rhd^{q}_{\leq 3} done)$$

There are other applications in various domains such as hardware analysis, networking protocols, the medical industry, aeronautics, and biology:

- Clock Difts Correction Mechanism: The temporal accuracy of information in a DRTS depends on the accuracy of the global time. This precision of the distributed real time depends on the fluctuation of the message transmission delay, the clock drifts and the global clock mechanism. A distributed real-time communication system must guarantee message transmission within a constant transmission delay with bounded variation and clock drift. The duration of message transmission over the network depends on the assumptions made about the network traffic. The focus in this scenario is the generation of a fault-tolerant global time base of high precision in a DRTS with clock drifts (local clocks). Therefore, we can use DECA to model this scenario [KAH06].
- Global Positioning System: Satellite networks (geosynchronous) have always had some undeniable advantages over other networks: natural coverage and rapid deployment in the event of natural disasters, for example. The main applications of satellite communications remain maritime communications, disaster relief, Internet access in remote areas, and mobile communications. Satellite systems play a very important role in interconnecting heterogeneous networks to provide global coverage to users. To be fully integrated into the global network, satellite systems will be able to efficiently route the communication protocol, TCP/IP, and a wide variety of applications with the same level of performance as networks on land. Satellite systems often require fairly precise communication and independent relationship between spacecraft local clocks. In the traditional method of achieving communication, a ground station makes time offset measurements to the various spacecraft clocks and then updates the time and frequency of each satellite as needed. Although simple to implement, disadvantages of the traditional approach include the heavy workload on the ground station, the need for multiple ground stations for new satellites in different orbital locations, and unaccounted for delays in atmospheric propagation. In the new method for achieving communication, a ground station only needs to control the time and frequency of each satellite, and its workload, and thus the system's time-related operating costs, are kept to a minimum. Moreover, since inter-satellite communication is achieved without transmission through the ionosphere, atmospheric propagation delays cannot disturb the independent relationship between the local clocks of the satellites. Therefore, we can use DECA to model this scenario [Ash03].
- ZigBee Use For Wireless Network: ZigBee is a transmission standard for machine to machine wireless network communication. ZigBee uses the IEEE

802.15.4 PHY and MAC layers. Data rates are relatively low, between 20 and 250 kbps. ZigBee operates primarily in the 2.4 GHz frequency band with 16 channels. Its range is tens of meters. A ZigBee network can contain up to 254 nodes per cell, in addition to a node that manages it, called a coordinator. The IEEE 802.15.4 standard supports two modes of operation managed by the network coordinator: unmarked and marked mode. The super frame periodically issued by the network coordinator consists of an active portion, during which the coordinator interacts with its child nodes, and an inactive portion, during which all nodes enter a low-power mode. In the wireless network, the problem of distributed real-time becomes much more difficult. Nodes may be far from any wired infrastructure, and environmental factors make sensors susceptible to failure and clock drift. In fact, there has been a lot of work in this area, such as sensor networks, the medical industry, and aeronautics. The problem involves two or more processors whose clocks are not synchronized. That is, when a processor starts, each clock starts counting up from 0; however, the processors may start at different times. The difference between the times at which the processors start is bounded by some positive real parameter. Therefore, we can use DECA to model this scenario [ERGM09].

6.5 Strengths and Weaknesses of the Formalisms

This section aims at showing the strengths and weaknesses of the two formalisms presented in this chapter.

6.5.1 Strengths

DECA are closed under all boolean operations, and their timed language inclusion problem is decidable (more precisely, PSPACE-complete), allowing stepwise refinement. Existential and universal timed languages are decidable. DECTL is a real-time logic with multiple observers, each with its time evolution. This logic can be model-checked by translating a DECTL formula into a DECA automaton. DECTL is also decidable (more precisely PSPACE-complete).

6.5.2 Weaknesses

ECA is a widely recognized subset of TA, known for its remarkable theoretical properties, in particular its determinizability (DECA inherits the same theoretical properties of ECA). However, despite the extensive research and implementation of TA, the concrete implementation of ECA remains challenging. This is due to the inherent complexity of adapting zone-based extrapolation techniques, which are crucial in the context of TA, to effectively handle the intricacies of ECA. To address this issue, Geeraerts et al., [GRS11] [GRS14] proposed a promising solution based on the use of a zone-based extrapolation technique. Geeraerts et al., [GRS11] [GRS14] showed the fundamental differences between predicting clocks and usual clocks. In particular, they showed that there is no finite time-abstract bisimulation for ECA in general, thus emphasizing the unique nature of predicting clocks in this context. In the context of TA, it is worth noting that the concept of region equivalence forms a finite time-abstract bisimulation. Consequently, a translation from ECA to TA has been proposed by Alur et al. [AFH94]. However, it should be acknowledged that this translation may not be efficient, as it can lead to an exponential increase in the number of clocks and states [AFH94].

Indeed, recent studies have highlighted that simulation-based techniques [GMS19] [BGH$^+$22] exhibit superior effectiveness [Bou04b] and efficiency [GMS19] [GMS20] [GMS18] [BGH$^+$22] in comparison to zone-based extrapolation techniques for reachability analysis in ECA. These findings emphasize the advantages of simulation-based techniques when it comes to verifying reachability in ECA. Additionally, in a recent study by Akshay et al. [AGGS22], a zone-based reachability algorithm was proposed specifically for diagonal-free ECA. However, it is important to note that no implementation details were provided alongside their proposal. Therefore, since DECA is an extension of ECA and RECA, the same difficulty of region equivalence and finite time-abstract bisimulation will be present in DECA.

Another issue up to now is that there are no tools supporting ECA (and RECA) with continuous semantics for modeling and verifying DRTS, despite clear practical interests [RDS$^+$15]. Several papers [MNP06] [BRS13] have emphasized the need for tools with continuous semantics. The absence of tools for continuous semantics can be attributed to a limited practical understanding and utilization of formalisms that deal with continuous semantics, such as MITL, EventClockTL, Signal Temporal Logic (STL) [MVS$^+$18]. There are already translations from MITL to TA in the literature [MNP06] [BRS13] [BGHM17b]. However, the translations are complex, in spite of some simplified constructions, such as [MNP06] [BRS13] [BGHM17b].

Another issue present in DECA and DECTL (also ECA, MTIL, RECA, and EventClockTL) is the choice of a continuous semantics (as opposed to a pointwise semantics) for modeling and verifying DRTS. Although pointwise semantics offer simplicity and are currently more prevalent, largely due to the popularity of TA, there is an ongoing argument [HR04] [HRS98] that continuous semantics provides a more faithful representation of time. Indeed, continuous semantics has been widely adopted in several domains, such as hybrid systems and synthetic biology [RDS$^+$15]. Beyond practical considerations, the distinction between these two semantics is significant because it changes the expressive power of logic. This highlights the importance of exploring and using continuous semantics formalisms to capture the subtleties of time more accurately and comprehensively. However, interpreting and evaluating formulas over continuous time intervals can be computationally intensive, requiring sophisticated algorithms, numerical techniques, and efficient data structures to handle the large amounts of temporal data. Developing robust and efficient tools that can handle the complexity of continuous semantics can require considerable effort and expertise. One way to avoid the complexity involved in implementing a tool based on DECA and DECTL is to perform a translation from DECTL to TA (or signal automata), such as [BGHM17b]. Brihaye et al., [BGHM17b] [BGHM17a] proposed a translation from MITL with continuous semantics to TA. This translation can be used as the basis for developing a model checking tool that translates a MITL formula into one or more TA, integrates these automata with a network of TA, and performs an emptiness check to verify the desired properties. However, it is important to note that there is currently no existing automated tool that fully supports this workflow. However, due to the high cost of translating MITL into TA and the need to construct its entire deterministic region automaton, this algorithm is unlikely to be amenable to implementation [BEG$^+$16].

6.6 Wrap up

This chapter introduces Recursive Event Clock Automata (RECA) with such distributed (a.k.a. independent) clocks, giving rise to Distributed Recursive Event Clock Automata (DECA). We have shown that DECA are determinizable, i.e. closed under complementation, and thus that their language inclusion problem is decidable (more precisely, PSPACE-complete). We also showed the decidability and regularity of their universal languages. In the second section, we have studied the corresponding timed languages of DTA and icTA. In third section, we extended EventClockTL with distributed clocks. This gives us the (Recursive) Distributed Event Clock Temporal Logic (DECTL), which we have shown to be PSPACE-complete. In the fourth section, we have shown some examples of distributed real-time models built over DECA and DECTL. Finally, in the fifth section, we have shown the strengths and weaknesses of DECA and DECTL.

Chapter 7

Multiple Independent Clocks

Due to implementation limitations encountered with the zone-based techniques and model checker tool for DECA and DECTL, as discussed in Section 6.5, in this chapter, we will define the formalisms of (Distributed) Multi-Timed Automata (MTA) and Distributed Timed Modal Logic (ML$_{v}$) in the context of TT for DRTS. MTA are a variant of TA inspired by Distributed Timed Automata (DTA) and TA with Independent Clocks (icTA) and proposed by [Kri99, ABG^{+}08, OLS11] to model DRTS. Indeed, DRTS involves multiple interconnected real-time processes where each process uses its local clocks running at its rate. The local time rates associated with each process may cause actions generated at the same instant to have different timestamps at each local clock. Because of the complex interactions between local timing constraints and the distributed behavior of DRTS, specifying and analyzing the correctness of these systems is costly and arduous. Furthermore, it is well known that TA and their variants (DTA and icTA) are neither determinizable nor complementable and their inclusion problem is undecidable [AD94]. In [OLS11] was defined a determinizable formalism for modeling non-deterministic DRTS. ML$_{v}$ is a logic that allows the specification of both the local behavior of the components and multiple local times in timed temporal specifications, facilitating the identification of multiple interactions between distributed processes.ML_ν describes properties of states in a Multi-Timed Labelled Transition Systems (MLTS) over a set of actions. We define its syntax and multi-timed semantics, and illustrate the most important features of ML_ν through the specification of DRTS. The Satisfiability and validity problems for ML_ν are decidable.

This chapter is structured as follows. In the first section 7.1, we propose to model DRTS with Multi-timed Automata (MTA), an extension of TA and icTA, whose execution traces can be modeled as sequences of pairs, where each pair contains an action and a tuple of timestamps. Thus, each action has its tuple of local time of occurrence for each process belonging to the modeling DRTS. We then propose to extend the theory of Timed Labeled Transition Systems (TLTS) to Multi-Timed Labeled Transition Systems (MLTS) and relate it to our alternative operational semantics for MTA. In the second section 7.2, we propose to reconsider the notion of timed bisimulation on these automata, leading to multi-timed bisimulation. In the third section 7.3 we have shown the parallel composition between MTA. In the fourth section 7.4, we prove its decidability and present an EXPTIME algorithm for deciding whether two MTA are multi-timed bisimilar. In the fifth section 7.5, we propose ML_ν, an extension of L_ν that relies on a distributed semantics for Timed Automata (TA): instead of considering uniform clocks over the distributed systems, we let time vary independently in each TA. We define the syntax and semantics of ML_ν over executions of MLTS with such semantics, and we show that its model checking problem against ML_ν is EXPTIME-complete. In the sixth section 7.6, we present some scenarios of distributed real-time systems for MTA and ML_ν. Finally, in the eighth section 7.7, we show the strengths and weaknesses of MTA and ML_ν.

7.1 An Alternative Semantics for DRTS

TA and MTL have been used to specify and model RTS, but these formalisms are not expressive enough to specify and analyze the correctness of DRTS for two main reasons. First, TA assumes perfectly synchronized clocks, while DRTS uses local clocks. Second, the standard semantics of TA is sequential and based on a Timed Labelled Transition System (TLTS), i.e. a run of a TA is given by a sequence of actions and timestamps, while local clocks give rise to multiple timestamps. However, a distributed semantics for TA and network of TA has been introduced in [BC13], but the associated semantics remains in the classical setting of perfect clocks evolving at the same rate.

Here we propose Multi-Timed Automata (MTA), a variant of TA inspired by Distributed Timed Automata (DTA) and Timed Automata with Independent Clocks (icTA), and proposed by [Kri99, ABG$^+$08, OLS11] to model DRTS. In fact, DRTS involves multiple interconnected real-time processes, each of which uses its own local clocks running at its own rate. The local time rates associated with each process can cause actions generated at the same time to have different timestamps at each local clock. An execution trace in our MTA is denoted by a sequence of pairs (i.e., an action and a tuple of timestamps). Thus, in each execution trace, each action has its own tuple of the local time of occurrence for each clock. Thus, this tuple of timestamps allows representing the distribution of the local time. It then becomes possible to analyze the local behavior of the components independently. We also propose Multi-Timed Labelled Transition Systems (MLTS), an extension of the classical TLTS to include the notion of multiple local times, and we propose efficient algorithms for bisimulation using partition refinement techniques [PT87].

Furthermore, using our alternative semantics of MTA allows us to describe the behavior of local clocks. We also extend the classical theory of timed bisimulation with the notion of multi-timed bisimulation. We present two algorithms: (1) a forward reachability algorithm for the lockstep composition of two MTA, which will help us reduce the state space exploration of our second algorithm, and (2) a decision algorithm for multi-timed bisimulation using zones [BY04]. Multi-timed bisimulation cannot be computed with the standard partition refinement algorithm [PT87]. Instead, our algorithm successively refines a set of zones such that each zone eventually contains only multi-timed bisimilar state pairs. Furthermore, we show that our algorithm is EXPTIME-complete. Since TA is a special case of MTA, and since timed bisimulation over TA [WL97] [TY01] can be regarded as a special case of multi-timed bisimulation, our decision algorithm also applies to this classical case.

7.1.1 Rates

As explained above, TA assumes that clocks are perfectly synchronous, which cannot always be guaranteed in distributed real-time applications where clocks may drift (count time at different rates) due to environmental conditions such as temperature, humidity, pressure, and aging. Let $Proc$ be a finite and non-empty set of processes. Often, the behavior and properties of DRTS depend on the local time rates at the processes. Each process may have a different local time evolution. To model this, we assume that the local time evolution can be represented as a function mapping the reference time to the local time (see definition of 57).

7.1.2 Multi-Timed Actions

An execution of a DRTS can be described by a multi-timed word, which is a sequence of actions with multiple timestamps indicating that they should be executed at different times. A multi-timed language describes a set of executions as a set of multi-timed words.

Let $Proc$ be a non-empty set of processes, then, we denote by $\mathbb{R}_{\geq 0}^{Proc}$ the set of functions from $Proc$ to $\mathbb{R}_{\geq 0}$, that we call tuples. A tuple $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$ is smaller than $\vec{d}'$, noted, $\vec{d} < \vec{d}'$ iff $\forall p \in Proc\ \vec{d}_p \leq \vec{d}'_p$ and $\exists q \in Proc\ \vec{d}_q < \vec{d}'_q$. A Monotone Sequence of Tuples (MST) is a sequence $\vec{d} = \vec{d}_1 \vec{d}_2 \cdots \vec{d}_n$ of tuples of $\mathbb{R}_{\geq 0}^{Proc}$ where : $\forall j \in 1 \cdots n-1$, $\vec{d}_j < \vec{d}_{j+1}$ or $\vec{d}_j = \vec{d}_{j+1}$.

Definition 71. *A multi-timed word on Σ is a pair $\theta = (\sigma, \vec{d})$ where $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ is a finite word $\sigma \in \Sigma^*$, and $\vec{d} = \vec{d}_1\vec{d}_2 \ldots \vec{d}_n$ is a MST of the same length. This is the analog*

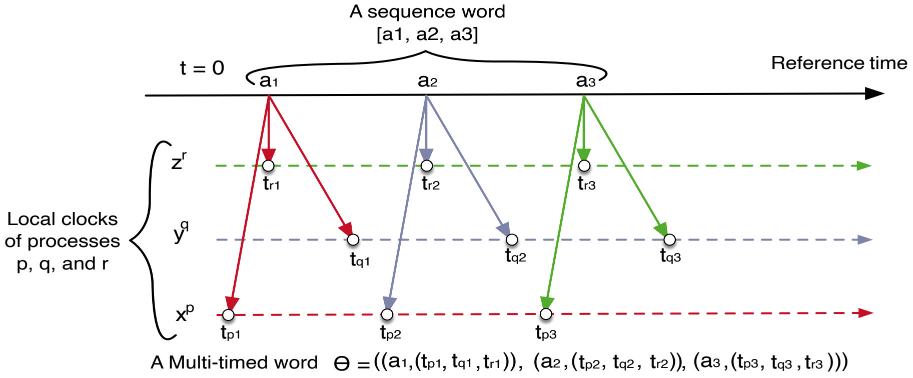


Figure 7.1: Local clocks and reference time

of a timed word [AD94]. A multi-timed word can equivalently be seen as a sequence of pairs in $\Sigma \times \mathbb{R}^X_{\geq 0}$.

Example 20. *In Figure 7.1, we consider three processes (i.e.* $Proc = \{p, q, r\}$*) with one clock each (i.e.,* $\pi(x^p) = p$, $\pi(y^q) = q$, $\pi(z^r) = r$*). The top line represents the reference time at which three actions* $\{a_1, a_2, a_3\}$ *occur at different times. The dotted lines below capture the time elapsed as perceived by each process: a white dot represents the local instant at which the action is perceived by a given process. For example,* a_1 *is perceived at* t_{r1} *by r and* t_{p1} *by p. Since local times are monotone, strictly increasing sequences of instants, the way these local times are perceived is always similar (as conveyed by the uniform way actions are related to local instants by simple arrows). For the three actions above, we get a multi-timed word* $\theta = ((a_1, t_{p1}, t_{q1}, t_{z1})(a_2, t_{p2}, t_{q2}, t_{r2})(a_3, t_{p3}, t_{q3}, t_{r3}))$ *where* $(t_{pi},\ t_{qi},\ t_{ri}) \in \mathbb{R}^{Proc}_{\geq 0}$ *for all* $i \in \{1,2,3\}$*, are related to the reference time according to the arrows in Figure 7.1.*

7.1.3 Multi-Timed Labeled Transition Systems (MLTS)

Here we present our formalism for reasoning about behavioral and temporal properties of DRTS using operational semantics based on TLTS. To represent the temporal properties of DRTS, we need to extend TLTS (i.e., an *alternative semantics*) with the ability to represent local elapsed time. Therefore, we propose an alternative semantics to the standard semantics for TLTS: specifically, a run in our *alternative semantics* is denoted by a sequence of pairs (i.e., an action and a tuple of timestamps). Thus, in each run, each action has its own tuple of local time of occurrence for each local clock in such a modeling DRTS. Therefore, this tuple of timestamps allows to represent the distribution of the local time for each local clock belonging to the modeling DRTS. Our *alternative semantics* is defined in terms of runs that record the state and local clock values at each transition point traversed during the consumption of a *multi-timed* word. Instead of observing actions at a global time, a multi-timed word records the time seen by each local clock. The actions that occur over a set of local clocks can be ordered by their time of occurrence (i.e., their timestamps).

Definition 72 (Multi-Timed Labelled Transition System). *A Multi-Timed Labelled Transition System (MLTS) over a set of processes $Proc$ is a tuple $\mathscr{M} = (Q, Q_F, q_0, \Sigma, \rightarrow_{mlts})$ such that:*

(i) Q is a set of states,

(ii) $q_0 \in Q$ is the initial state,

(iii) Σ is a finite alphabet,

(iv) Q_F is a set of final states,

(v) $\rightarrow_{mlts} \subseteq Q \times (\Sigma \uplus \mathbb{R}_{\geq 0}^{Proc}) \times Q$ is a set of transitions.

The transitions from state to state of a MLTS are noted in the following way:

(i) A discrete transition (q, a, q') is denoted $q \xrightarrow{a} q'$, if $a \in \Sigma$,

(ii) A delay transition $(q, \vec{d}, q')$ is denoted $q \xrightarrow{\vec{d}} q'$, if $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$.

A path (or run) of $\mathscr{M}$ can be defined as a finite sequence of moves, where discrete and delay transitions alternate: $\rho = q_0 \xrightarrow{\vec{d}_1} q'_0 \xrightarrow{a_1} q_1 \xrightarrow{\vec{d}_2} \ldots q_{n-1} \xrightarrow{\vec{d}_n} q'_{n-1} \xrightarrow{a_n} q_n$, where $\forall\ 0 \leq i \leq n$, $q_i \in Q$, $\forall 1 \leq j \leq n$, $\vec{d}_j \in \mathbb{R}_{\geq 0}^{Proc}$, $q'_j \in Q$ and $a_j \in \Sigma$. A path is initial if it starts in q_0. A path is accepting if it starts in initial state q_0 and ends in a final state $q_f \in Q_F$. The *multi-timed word* of ρ is $\theta = ((a_1, \vec{t}_1), (a_2, \vec{t}_2) \ldots, (a_n, \vec{t}_n))$, where $\vec{t}_i = \sum_{j=1}^{i} \vec{d}_j$. A multi-timed word θ is *accepted* by $\mathscr{M}$, called multi-timed language, if and only if there exists an accepting run whose multi-timed word is θ. The *language* of $\mathscr{M}$, denoted $\mathscr{L}(\mathscr{M})$, is defined as the set of all finite multi-timed words accepted by $\mathscr{M}$. Note that MLTS are a proper generalization of TLTS: each TLTS can be seen as a MLTS with a single process.

Furthermore, our MLTS must satisfy the following multi-timed requirements:

(i) Multi-determinism: for all q, q', $q'' \in Q$ and for all $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$, if $q \xrightarrow{\vec{d}} q'$ and $q \xrightarrow{\vec{d}} q''$ then $q' = q''$.

(ii) Multi-$\bar{0}$-Delay: Let $(f_{\bar{0}} : Proc \mapsto 0) \in \mathbb{R}_{\geq 0}^{Proc}$ be the hull-tuple such that for all q, $q' \in Q$, $q \xrightarrow{f_{\bar{0}}} q' \equiv q = q'$.

(iii) Multi-Additivity: for all q, q', $q'' \in Q$ and for any $\vec{d}$, $\vec{d'} \in \mathbb{R}_{\geq 0}^{Proc}$, if $q \xrightarrow{\vec{d}} q'$ and $q' \xrightarrow{\vec{d'}} q''$ then $q \xrightarrow{\vec{d}+\vec{d'}} q''$.

(iv) Multi-Continuity: for all q, $q' \in Q$ and $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$, if $q \xrightarrow{\vec{d}} q'$ then there exists $\tau \in Rates$, such that for a reference time $t = 1$, $\tau(t) = \vec{d}$ and for all $t \leq 1$, there exists q'' such that $q \xrightarrow{\tau(t)} q''$ and $q'' \xrightarrow{\vec{d}-\tau(t)} q'$.

Example 21. *For example, consider the two transition systems in Figure 7.2: (a) a MLTS on the left ($\mathscr{M}_1$) and (b) two TLTS on the right ($\mathscr{M}_2$ and $\mathscr{M}_3$) with the finite input alphabet $\Sigma = \{a, b, c\}$. In brief, $\mathscr{M}_2$ and $\mathscr{M}_3$ could be considered as the projection of $\mathscr{M}_1$ on process 1 and 2.*

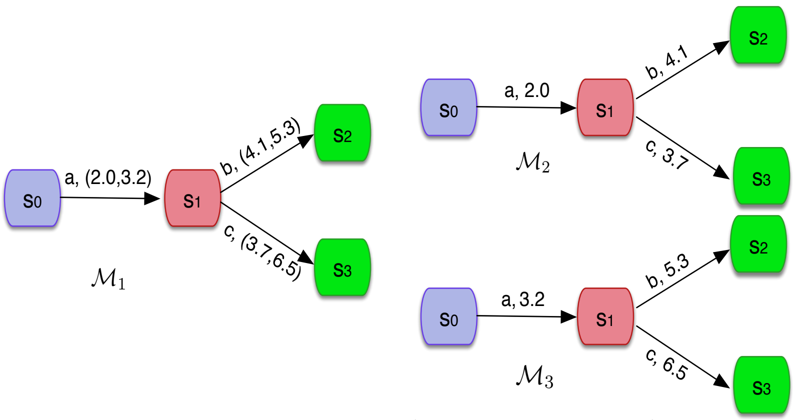


Figure 7.2: (a) A MTLS ($\mathscr{M}_1$). (b) Two TLTS ($\mathscr{M}_2$) and ($\mathscr{M}_3$)

7.1.4 Multi-timed Automata

Distributed Timed Automata (DTA) [Kri99, ABG$^+$08] consist of several local timed automata called processes. Each process has its own clocks. The clocks of the same process evolve synchronously, but independently of the clocks of other processes. In [ABG$^+$08], DTA are not studied much. Instead, their product is computed first, resulting in the class of icTA. Inspired by icTA and DTA [Kri99, ABG$^+$08], we introduce Multi-timed Automata (MTA) to model DRTS. The semantics of a multi-timed automaton is given by our MLTS.

Definition 73 (MTA)**.** *A MTA is a pair $\mathscr{A} = (\mathscr{B}, \pi)$ over Proc where :*

(i) $\mathscr{B} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F)$ is a TA,

(ii) $\pi : X \rightarrow Proc$ maps each clock to a process.

Definition 74. *Given $\pi : X \rightarrow Proc$, a clock valuation $\nu : X \rightarrow \mathbb{R}_{\geq 0}$ and $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$: the valuation $\nu +_\pi \vec{d}$ is defined by $(\nu +_\pi \vec{d})(x) = \nu(x) + \vec{d}_{\pi(x)}$ for all $x \in X$.*

Definition 75 (**Semantics of Multi-timed Automata** (MTA))**.** *Given a MTA $\mathscr{A} = (S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F, \pi)$ over Proc and $\tau \in Rates$, the alternative semantics of $\mathscr{A}$ is given by a MLTS over Proc, denoted by MLTS($\mathscr{A}$, τ) = (Q, q_0, Q_F, Σ,$\rightarrow_{mlts}$). The set of states Q consists of triples composed of a location, a clock valuation and lastly the reference time: $Q = \{(s, \nu, t) \in S \times \mathbb{R}_{\geq 0}^X \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s)\}$. The set of final states Q_F consists of triples $\{(s_f, \nu, t) \in F \times \mathbb{R}_{\geq 0}^X \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s_f)\}$. The starting state is $q_0 = (s_0, \nu_0, 0)$, where ν_0 is the valuation that initializes all the clocks to zero. Σ is the alphabet of $\mathscr{A}$. The transition relation $\rightarrow_{mlts}$ is defined by:*

(i) A transition $(q_i, \vec{d}, q_i')$ is denoted $q_i \xrightarrow{\vec{d}} q_i'$, and is called a *delay transition*, where $q_i = (s_i, \nu_i, t_i)$, $q_i' = (s_i, \nu_i +_\pi \vec{d}, t_{i+1})$, $\vec{d} = \tau(t_{i+1}) - \tau(t_i)$ and $\forall t \in [t_i, t_{i+1}]$: $\nu_i +_\pi (\tau(t) - \tau(t_i)) \models Inv(s_i)$.

(ii) A transition (q_i, a, q_{i+1}) is denoted $q_i \xrightarrow{a} q_{i+1}$, and is called a *discrete transition*, where $q_i = (s_i, \nu_i, t_i)$, $q_{i+1} = (s_{i+1}, \nu_{i+1}, t_{i+1})$, $a \in \Sigma$, there exists a transition

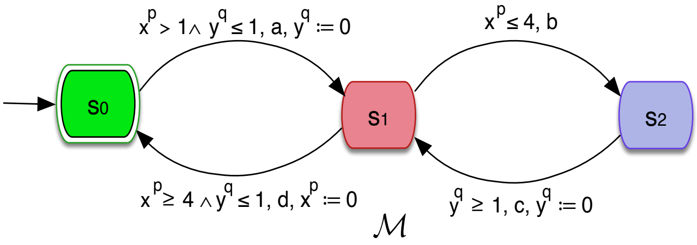


Figure 7.3: A Multi-timed Automaton $\mathscr{M}$

$(s_i, a, \phi, Y, s_{i+1}) \in \rightarrow_{\mathsf{ta}}$, such that $\nu_i \models \phi$, $\nu_{i+1} = \nu_i[Y \leftarrow 0]$, $\nu_{i+1} \models Inv(s_{i+1})$, $t_i = t_{i+1}$.

A path of $\mathscr{A}$ for $\tau \in Rates$ with its *alternative* semantics is an initial path in $\mathsf{MLTS}(\mathscr{A}, \tau)$ where discrete and continuous transition alternate. A multi-timed word is accepted by $\mathscr{A}$ for $\tau \in Rates$ iff it is accepted by $\mathsf{MLTS}(\mathscr{A}, \tau)$. The multi-timed language accepted by $\mathscr{A}$ for τ is denoted as $\mathscr{L}(\mathscr{A}, \tau)$.

Example 22. *Figure 7.3 above shows a MTA $\mathscr{M}$ with the finite alphabet $\Sigma = \{a, b, c, d\}$, the set of processes $Proc = \{p, q\}$, the set of clocks $X = \{x^p, y^q\}$ and $\tau = (2t, t)$ (i.e., $\tau_p(t) = 2t$ and $\tau_q(t) = t$). A run of $\mathscr{M}$ on a multi-timed word $\theta = ((a, (2.0, 1.0))(b, (3.0, 1.5))$ $(c, (4.2, 2.1))(d, (6.0, 3.0)))$ is given by* $(s_0, [x^p = 0.0, y^q = 0.0], 0.0) \xrightarrow{(2.0,1.0)} (s_0, [x^p = 2.0, y^q = 1.0], 1.0) \xrightarrow{a} (s_1, [x^p = 2.0, y^q = 0.0], 1.0) \xrightarrow{(1.0,0.5)} (s_1, [x^p = 3.0, y^q = 0.5], 1.5) \xrightarrow{b} (s_2, [x^p = 3.0, y^q = 0.5], 1.5) \xrightarrow{(1.2,0.6)} (s_2, [x^p = 4.2, y^q = 1.1], 2.1) \xrightarrow{c} (s_1, [x^p = 4.2, y^q = 0.0], 2.1) \xrightarrow{(1.8,0.9)} (S_1, [x^p = 6.0, y^q = 0.9], 3.0) \xrightarrow{d} (s_0, [x^p = 0.0, y^q = 0.9], 3.0)$.

7.2 Multi-timed Bisimulation

In the earlier paper [Cer93], the author provided timed equivalences in which equivalent processes must completely match in their timing properties as well as in their functional behavior. However, the timing properties of any two processes may not be completely equivalent, because each local process may not compute at the same speed, and local clocks may not run at the same rate. Then it would be desirable to treat two different processes as equivalent only if their behaviors are completely matched. Therefore, we develop such equivalences by extending the classical definition of timed bisimulation [Cer93] towards our alternative semantics. Our motivation for extending the classical definition of timed bisimulation is twofold: first, efficient algorithms for checking timed and time-abstract bisimulation have been discovered [Cer93] [WL97]. However, these algorithms are based on sequential semantics (i.e., TLTS and TA). Second, verifying the preservation of distributed temporal behavior in DRTS could be used to cope with the combinatorial explosion of the size of the model.

7.2.0.1 Strong Multi-timed Bisimulation

Let $\mathscr{M}_1$ and $\mathscr{M}_2$ be two MLTS over the same set of actions Σ and processes $Proc$. Let $Q_{\mathscr{M}_1}$ (resp., $Q_{\mathscr{M}_2}$) be the set of states of $\mathscr{M}_1$ (resp., $\mathscr{M}_2$). Let $\mathscr{R}$ be a binary relation over $Q_{\mathscr{M}_1} \times Q_{\mathscr{M}_2}$. We say that $\mathscr{R}$ is a strong multi-timed bisimulation whenever the following transfer property holds (note that technically this is simply strong bisimulation over $\Sigma \uplus \mathbb{R}_{\geq 0}^{Proc}$):

Definition 76 (**Strong Multi-timed Bisimulation**)**.** *A strong multi-timed bisimulation over MLTS $\mathscr{M}_1, \mathscr{M}_2$ is a binary relation $\mathscr{R} \subseteq Q_{\mathscr{M}_1} \times Q_{\mathscr{M}_2}$ such that, whenever $q_{\mathscr{M}_1} \mathscr{R} q_{\mathscr{M}_2}$, the following holds:*

(i) For every $a \in \Sigma$ and for every discrete transition $q_{\mathscr{M}_1} \xrightarrow{a}_{\mathscr{M}_1} q'_{\mathscr{M}_1}$, there exists a matching discrete transition $q_{\mathscr{M}_2} \xrightarrow{a}_{\mathscr{M}_2} q'_{\mathscr{M}_2}$ such that $q'_{\mathscr{M}_1} \mathscr{R} q'_{\mathscr{M}_2}$ and vice versa.

(ii) For every $\vec{d} = (d_1, \ldots, d_n) \in \mathbb{R}_{\geq 0}^{Proc}$, for every delay transition $q_{\mathscr{M}_1} \xrightarrow{\vec{d}}_{\mathscr{M}_1} q'_{\mathscr{M}_1}$, there exists a matching delay transition $q_{\mathscr{M}_2} \xrightarrow{\vec{d}}_{\mathscr{M}_2} q'_{\mathscr{M}_2}$ such that $q'_{\mathscr{M}_1} \mathscr{R} q'_{\mathscr{M}_2}$ and vice versa.

Two states $q_{\mathscr{M}_1}$ and $q_{\mathscr{M}_2}$ are multi-timed bisimilar, written $q_{\mathscr{M}_1} \approx q_{\mathscr{M}_2}$, iff there is a multi-timed bisimulation $\mathscr{R}$ such that $q_{\mathscr{M}_1} \mathscr{R} q_{\mathscr{M}_2}$. $\mathscr{M}_1$ and $\mathscr{M}_2$ are multi-timed bisimilar, written $\mathscr{M}_1 \approx \mathscr{M}_2$, if there exists a multi-timed bisimulation relation $\mathscr{R}$ over $\mathscr{M}_1$ and $\mathscr{M}_2$ containing the pair of initial states. Furthermore, for all $q_{\mathscr{M}_1} \; \mathscr{R} \; q_{\mathscr{M}_2}$, if $q_{\mathscr{M}_1} \in Q^F_{\mathscr{M}_1}$ then $q_{\mathscr{M}_2} \in Q^F_{\mathscr{M}_2}$.

As a consequence of Definition 75, the notion of multi-timed bisimulation extends to MTA and we have the following definition:

Definition 77 (Multi-timed Bisimilar)**.** *Let $\mathscr{A}$ and $\mathscr{B}$ be two MTA. We say the automata $\mathscr{A}$ and $\mathscr{B}$ are multi-timed bisimilar, denoted $\mathscr{A} \approx \mathscr{B}$, iff $\forall \; \tau \in Rates$ MLTS($\mathscr{A}$, τ) $\approx$ MLTS($\mathscr{B}$, τ).*

When there is only one process, the multi-timed bisimulation is the usual timed bisimulation.

Example 23. *Consider the two MTA $\mathscr{A}_p$ and $\mathscr{A}_q$ in Figure 7.4 with the alphabet $\Sigma = \{a\}$, the set of processes $Proc = \{p, q\}$, the set of clocks $X = \{x^p, y^q\}$ and $\tau = (t^2, 3t)$ i.e. $\tau_p(t) = t^2$ and $\tau_q(t) = 3t$. $\mathscr{A}_p$ performs nondeterministically the transition with the guard $x^p \leq 2$, the action a, resets clock x^p to 0 and enters location s_1. Similarly, $\mathscr{A}_q$ is the same but with y^q. We will show that these MTA are not multi-timed bisimilar (Definition 76) even if their underlying TA are bisimilar (and even isomorphic): We have $(s_0, [x^p = 0.0], 0.0)$ in MLTS($\mathscr{A}_p, \tau$) and $(r_0, [y^q = 0.0], 0.0)$ in MLTS($\mathscr{A}_q, \tau$). Then, since $\mathscr{A}_p$ can run the delay transition $(s_0, [x^p = 0.0], 0.0) \xrightarrow{(1,3)} (s_0, [x^p = 1.0], 1.0)$ and then $(s_0, [x^p = 0.0], 0.0) \xrightarrow{a} (s_1, [x^p = 1.0], 1.0)$. $\mathscr{A}_q$ can only match this transition with $(r_0, [y^q = 0.0], 0.0) \xrightarrow{(1,3)} (r_0, [y^q = 3.0], 1.0)$, where MLTS($\mathscr{A}_q, \tau_q$) cannot fire a.*

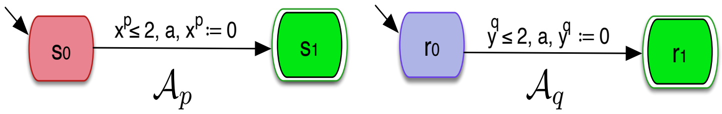


Figure 7.4: Two multi-timed automata $\mathscr{A}_p$ and $\mathscr{A}_q$

7.3 Parallel Composition of MTA

Normally, a DRTS consists of several processes that can be modeled independently. Their interaction is determined by a parallel composition operator (denoted by ∥) that merges them into a single process (i.e., in our case, a multi-timed automaton). MTA can interact in two ways: through synchronous discrete transitions and through asynchronous delay transitions (i.e., time is local to each automaton). Parallel composition can also be defined at the MLTS level.

Definition 78 (Composition of MLTS). *Let $\mathscr{M}_1 = (Q_{\mathscr{M}_1}, Q^F_{\mathscr{M}_1}, q^0_{\mathscr{M}_1}, \Sigma_{\mathscr{M}_1}, \rightarrow_{\mathscr{M}_1})$ and $\mathscr{M}_2 = (Q_{\mathscr{M}_2}, Q^F_{\mathscr{M}_1}, q^0_{\mathscr{M}_2}, \Sigma_{\mathscr{M}_2}, \rightarrow_{\mathscr{M}_2})$ be two MLTS over the set of processes $Proc_{\mathscr{M}_1}$ and $Proc_{\mathscr{M}_2}$. The parallel composition $\mathscr{M}_1 \parallel \mathscr{M}_2$ over the set of processes $Proc = Proc_{\mathscr{M}_1} \cup Proc_{\mathscr{M}_2}$ is $\mathscr{M}_3 = (Q_{\mathscr{M}_3}, Q^F_{\mathscr{M}_3}, q^0_{\mathscr{M}_3}, \Sigma_{\mathscr{M}_3}, \rightarrow_{\mathscr{M}_3})$ where:*

(i) $Q_{\mathscr{M}_3} = Q_{\mathscr{M}_1} \times Q_{\mathscr{M}_2}$,

(ii) $Q^F_{\mathscr{M}_3} = Q^F_{\mathscr{M}_1} \times Q^F_{\mathscr{M}_2}$,

(iii) $q^0_{\mathscr{M}_3} = (q^0_{\mathscr{M}_1}, q^0_{\mathscr{M}_2})$,

(iv) $\Sigma_{\mathscr{M}_3} = \Sigma_{\mathscr{M}_1} \cup \Sigma_{\mathscr{M}_2}$,

(v) $\rightarrow_{\mathscr{M}_3} \subseteq Q_{\mathscr{M}_3} \times (\Sigma_{\mathscr{M}_3} \uplus \mathbb{R}^{Proc}_{\geq 0}) \times Q_{\mathscr{M}_3}$ *is the set of transitions given by:*

  (a) A discrete transition $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), a, (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2}))$ *is denoted* $(q_{\mathscr{M}_1}, q_{\mathscr{M}_2}) \xrightarrow{a} (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})$, *if* $a \in \Sigma_{\mathscr{M}_1} \cup \Sigma_{\mathscr{M}_2}$ *and* $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), a, (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})) \in \rightarrow_{\mathscr{M}_3}$ *and,*

  (b) A delay transition $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), \vec{d}, (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2}))$ *is denoted* $(q_{\mathscr{M}_1}, q_{\mathscr{M}_2}) \xrightarrow{\vec{d}} (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})$, *if* $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ *and* $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), \vec{d}, (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})) \in \rightarrow_{\mathscr{M}_3}$.

Definition 79 (Parallel Composition of MTA). *Let $\mathscr{A}$ and $\mathscr{B}$ be two TA. Let $\mathscr{A}' = (\mathscr{A}, \pi_{\mathscr{A}})$ and $\mathscr{B}' = (\mathscr{B}, \pi_{\mathscr{B}})$ be two MTA over the set of processes $Proc_{\mathscr{A}}$ and $Proc_{\mathscr{B}}$. We assume $Proc_{\mathscr{A}}$ and $Proc_{\mathscr{B}}$ are disjoint. The parallel composition of $\mathscr{A}'$ and $\mathscr{B}'$, written MTA $(\mathscr{A}' \parallel \mathscr{B}') = (\mathscr{A}, \pi_{\mathscr{A}}) \parallel (\mathscr{B}, \pi_{\mathscr{B}}) = ((\mathscr{A} \parallel \mathscr{B}), \pi_{\mathscr{A}} \cup \pi_{\mathscr{B}})$ over $Proc_{\mathscr{A}} \cup Proc_{\mathscr{B}}$ creates a new MTA $\mathscr{C}' = (\mathscr{C}, \pi)$ over $Proc$ where $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$, $\pi = \pi_{\mathscr{A}} \cup \pi_{\mathscr{B}}$, $Proc = Proc_{\mathscr{A}} \cup Proc_{\mathscr{B}}$ and $\mathscr{C} = (S_{\mathscr{C}}, s^0_{\mathscr{C}}, \Sigma_{\mathscr{C}}, X_{\mathscr{C}}, \rightarrow^{ta}_{\mathscr{C}}, Inv_{\mathscr{C}}, F_{\mathscr{C}})$ where:*

(i) $S_{\mathscr{C}} = S_{\mathscr{A}} \times S_{\mathscr{B}}$,

(ii) $s^0_{\mathscr{C}} = (s_{\mathscr{A}}, q_{\mathscr{B}})$,

(iii) $\Sigma_{\mathscr{C}} = \Sigma_{\mathscr{A}} \cup \Sigma_{\mathscr{B}}$,

(iv) $X_{\mathscr{C}} = X_{\mathscr{A}} \cup X_{\mathscr{B}}$,

(v) $\rightarrow^{ta}_{\mathscr{C}} \subseteq S_{\mathscr{C}} \times \Sigma_{\mathscr{C}} \times \Phi(X_{\mathscr{C}}) \times 2^{X_{\mathscr{C}}} \times S_{\mathscr{C}}$ *is the transition relation given by: for* $a \in \Sigma_{\mathscr{A}} \cup \Sigma_{\mathscr{B}}$, *if* $(s_{\mathscr{A}}, a, \phi_{\mathscr{A}}, Y_{\mathscr{A}}, s'_{\mathscr{A}}) \in \rightarrow^{ta}_{\mathscr{A}}$ *and* $(s_{\mathscr{B}}, a, \phi_{\mathscr{B}}, Y_{\mathscr{B}}, s'_{\mathscr{B}}) \in \rightarrow^{ta}_{\mathscr{B}}$ *then* $((s_{\mathscr{A}}, s_{\mathscr{B}}), a, \phi_{\mathscr{A}} \wedge \phi_{\mathscr{B}}, Y_{\mathscr{A}} \cup Y_{\mathscr{B}}, (s'_{\mathscr{A}}, s'_{\mathscr{B}})) \in \rightarrow^{ta}_{\mathscr{C}}$,

(vi) $Inv_{\mathscr{C}}(s_{\mathscr{A}}, s_{\mathscr{B}}) = Inv_{\mathscr{A}}(s_{\mathscr{A}}) \wedge Inv_{\mathscr{B}}(s_{\mathscr{B}})$ *for all* $s_{\mathscr{A}} \in S_{\mathscr{A}}$ *and* $s_{\mathscr{B}} \in S_{\mathscr{B}}$,

(vii) $F_{\mathscr{C}} = F_{\mathscr{A}} \times F_{\mathscr{B}}$,

Definition 80. *Let $\mathscr{A}$ and $\mathscr{B}$ be two MTA. For any valuations $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ over disjoint sets of clocks $X_{\mathscr{A}}$ and $X_{\mathscr{B}}$, there exists a unique valuation $v_{\mathscr{A}\|\mathscr{B}}$ over $X_{\mathscr{A}} \uplus X_{\mathscr{B}}$ with $v_{\mathscr{A}\|\mathscr{B}} = v_{\mathscr{A}} \uplus v_{\mathscr{B}}$. Given $v_{\mathscr{A}\|\mathscr{B}}(x)$, $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ can be recored as $v_{\mathscr{A}\|\mathscr{B}}(x)\rfloor_{X_{\mathscr{A}}}$ and $v_{\mathscr{A}\|\mathscr{B}}(x)\rfloor_{X_{\mathscr{B}}}$*

Let TA $\mathscr{A}$ and $\mathscr{B}$ be two TA and let $\mathscr{C} = (\mathscr{A}, \pi_{\mathscr{A}})$ and $\mathscr{D} = (\mathscr{B}, \pi_{\mathscr{B}})$ be two MTA over the set of processes $Proc_{\mathscr{C}}$ and $Proc_{\mathscr{D}}$. Given two states $q_{\mathscr{C}} = (s_{\mathscr{C}}, v_{\mathscr{C}}, t_{\mathscr{C}})$ of MLTS$(\mathscr{C}, \tau)$, and $q_{\mathscr{D}} = (s_{\mathscr{D}}, v_{\mathscr{D}}, t_{\mathscr{D}})$ of MLTS$(\mathscr{D}, \tau)$ for any $\tau \in Rates$, the unique state of MLTS$(\mathscr{C}, \tau) \parallel$ MLTS$(\mathscr{D}, \tau)$ corresponding to these states is written $((s_{\mathscr{C}}, s_{\mathscr{D}}), v_{\mathscr{C}\|\mathscr{D}}, t_{\mathscr{C}\|\mathscr{D}})$, where $v_{\mathscr{C}\|\mathscr{D}}(x) = v_{\mathscr{C}}(x)$ if $x \in X_{\mathscr{C}}$, and $v_{\mathscr{C}\|\mathscr{D}}(x) = v_{\mathscr{D}}(x)$ if $x \in X_{\mathscr{D}}$. The semantics of the parallel composition $((\mathscr{C} \parallel \mathscr{D}), \pi_{\mathscr{C}} \cup \pi_{\mathscr{D}})$ over $Proc_{\mathscr{C}} \cup Proc_{\mathscr{D}}$ will be given by means of a MLTS.

Definition 81 (**Semantics of the Parallel Composition**). *The MLTS generated by the parallel composition of $\mathscr{C}$ and $\mathscr{D}$ is a MLTS$(\mathscr{C}, \tau) \parallel$ MLTS$(\mathscr{D}, \tau) = (Q, Q_F, q_0, ((\Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}) \cup \mathbb{R}_{\geq 0}^{Proc}), \rightarrow_{mlts})$, where $Q = \{((s_{\mathscr{C}}, s_{\mathscr{D}}), v_{\mathscr{C}\|\mathscr{D}}, t_{\mathscr{C}\|\mathscr{D}}) \in ((S_{\mathscr{C}} \times S_{\mathscr{D}}) \times \mathbb{R}_{\geq 0}^{X} \times \mathbb{R}_{\geq 0}) \mid v_{\mathscr{C}\|\mathscr{D}} \models Inv_{\mathscr{C}}(s_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s_{\mathscr{D}})\}$ is the set of states. The set of final states Q_F consists of triples $\{((s_{\mathscr{C}}^{f}, s_{\mathscr{D}}^{f}), v_{\mathscr{C}\|\mathscr{D}}, t_{\mathscr{C}\|\mathscr{D}}) \in ((S_{\mathscr{C}}^{F} \times S_{\mathscr{D}}^{F}) \times \mathbb{R}_{\geq 0}^{X} \times \mathbb{R}_{\geq 0}) \mid v_{\mathscr{C}\|\mathscr{D}} \models Inv_{\mathscr{C}}(s_{\mathscr{C}}^{f}) \wedge Inv_{\mathscr{D}}(s_{\mathscr{D}}^{f})\}$. $q_0 = ((s_{\mathscr{C}}^{0}, s_{\mathscr{D}}^{0}), v_{\mathscr{C}\|\mathscr{D}}^{0}, 0)$, where $v_{\mathscr{C}\|\mathscr{D}}^{0}$ is the valuation that assigns 0 to all the clocks. Σ is the alphabet $\Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}$ and the transition relation $\rightarrow_{mlts}$ is defined by :*

(i) A delay transition $(q, \vec{d}, q^{'})$ is denoted $q \xrightarrow{\vec{d}} q^{'}$ where $q = ((s_{\mathscr{C}}, s_{\mathscr{D}}), v_{\mathscr{C}\|\mathscr{D}}, t_{\mathscr{C}\|\mathscr{D}}^{'})$, $q^{'} = ((s_{\mathscr{C}}, s_{\mathscr{D}}), v_{\mathscr{C}\|\mathscr{D}} +_{\pi} \vec{d}, t_{\mathscr{C}\|\mathscr{D}}^{''})$, $\vec{d} = \tau(t_{\mathscr{C}\|\mathscr{D}}^{''}) - \tau(t_{\mathscr{C}\|\mathscr{D}}^{'})$ and $\forall t \in [t_{\mathscr{C}\|\mathscr{D}}^{'}, t_{\mathscr{C}\|\mathscr{D}}^{''}]$: $v_{\mathscr{C}\|\mathscr{D}} +_{\pi} (\tau(t) - \tau(t_{\mathscr{C}\|\mathscr{D}}^{'})) \models Inv_{\mathscr{C}}(s_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s_{\mathscr{D}})$.

(ii) A discrete transition $(q, a, q^{'})$ is denoted $q \xrightarrow{a} q^{'}$ where $q = ((s_{\mathscr{C}}, s_{\mathscr{D}}), v_{\mathscr{C}\|\mathscr{D}}, t_{\mathscr{C}\|\mathscr{D}})$, $q^{'} = ((s_{\mathscr{C}}^{'}, s_{\mathscr{D}}^{'}), v_{\mathscr{C}\|\mathscr{D}}^{'}, t_{\mathscr{C}\|\mathscr{D}}^{'})$, $a \in \Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}$, there exists a transition $s_{\mathscr{C}} \xrightarrow{a, \phi_{\mathscr{C}}, Y_{\mathscr{C}}} s_{\mathscr{C}}^{'}$, $s_{\mathscr{D}} \xrightarrow{a, \phi_{\mathscr{D}}, Y_{\mathscr{D}}} s_{\mathscr{D}}^{'}$, such that $v_{\mathscr{C}\|\mathscr{D}} \models \phi_{\mathscr{C}} \wedge \phi_{\mathscr{D}}$, $v^{'} = v[(Y_{\mathscr{C}} \cup Y_{\mathscr{D}}) \rightarrow 0]$, $v_{\mathscr{C}\|\mathscr{D}}^{'} \models I_{\mathscr{C}}(s_{\mathscr{C}}^{'}) \wedge I_{\mathscr{D}}(s_{\mathscr{D}}^{'})$, $t^{'} = t^{''}$.

Example 24. *In Figure 7.5, two MTA $\mathscr{A}$ and $\mathscr{B}$, and their composition $\mathscr{A} \parallel \mathscr{B}$ are presented. The automaton $\mathscr{A}$, over the $Proc = \{p\}$ and local clock x^p can execute the actions a, b, with the transitions $s_0 \xrightarrow{a, true, x^p := 0}_{ta} s_1$, $s_1 \xrightarrow{b, x^p \geq 1, \emptyset}_{ta} s_0$. The possible transitions of the automaton $\mathscr{B}$ over the $Proc = \{q\}$ and local clock y^q are $r_0 \xrightarrow{b, true, y^q := 0}_{ta} r_1$, $r_1 \xrightarrow{c, y^q \geq 1, \emptyset}_{ta} r_0$. Both components execute in parallel and synchronize through the action b. The locations of the composition (automaton) are given as pairs (s_0, r_0), (s_0, r_1), (s_1, r_0), (s_1, r_1) whose elements corresponds to the locations of the automaton $\mathscr{A}$ and $\mathscr{B}$.*

Given two MTA $\mathscr{A} = (\mathscr{A}_1, \pi)$ and $\mathscr{B} = (\mathscr{A}_2, \pi)$, where $\mathscr{A}_1$ and $\mathscr{A}_2$ are two TA. The proof of the correctness of the MTA composition is based on proving the multitimed bisimulation between the semantics of the composition of the MTA $\mathscr{A}$ and $\mathscr{B}$. This means that we have to prove the parallel composition on the semantics for MTA (e.g., MLTS$(\mathscr{A}, \tau)$ and MLTS$(\mathscr{B}, \tau)$).

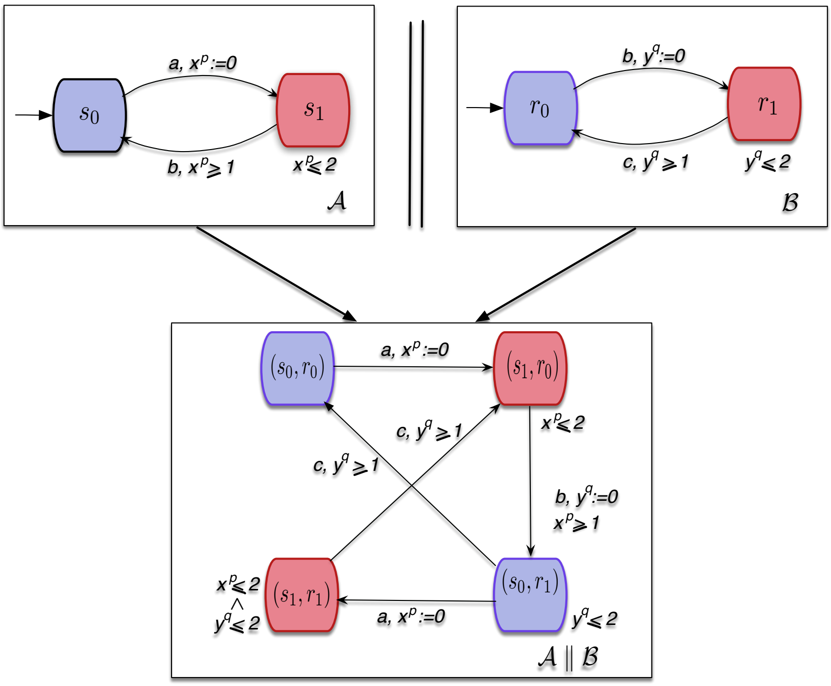


Figure 7.5: Parallel Composition of two MTA $\mathscr{A}$ and $\mathscr{B}$

Theorem 58. *Let $\mathscr{A}$ and $\mathscr{B}$ be two MTA, then for any $\tau \in Rates$, $MLTS(\mathscr{A},\tau) \parallel MLTS(\mathscr{B},\tau) = MLTS((\mathscr{A} \parallel \mathscr{B}),\tau)$.*

Proof. The proof consists in showing that each transition of the MLTS$((\mathscr{A},\tau) \parallel (\mathscr{B},\tau))$ can be found in MLTS$((\mathscr{A} \parallel \mathscr{B}),\tau)$ and vice versa. Let $\mathscr{R} = \{\, (((s_{\mathscr{A}},\ s_{\mathscr{B}}),\ \nu_{\mathscr{A}\parallel\mathscr{B}},\ t_{\mathscr{A}\parallel\mathscr{B}}),\ ((s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}}))) \mid \nu_{\mathscr{A}\parallel\mathscr{B}}(x) = \nu_{\mathscr{A}}(x) \uplus \nu_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$ and $t_{\mathscr{A}\parallel\mathscr{B}} = t_{\mathscr{A}} = t_{\mathscr{B}}\, \}$. Based on the MLTS and MTA parallel composition, there exists two types of transitions:

(i) Discrete transition: Let $q_{\mathscr{A}} = (s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})$ be two states of MLTS$(\mathscr{A},\tau)$ and MLTS$(\mathscr{B},\tau)$ respectively. A transition $((s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}),\ (s_{\mathscr{B}},\ \nu_{\mathscr{B}},\ t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}}, \nu'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, \nu'_{\mathscr{B}}, t'_{\mathscr{B}}))$ exists on MLTS$(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$ iff the transition $((s_{\mathscr{A}}, s_{\mathscr{B}}),\ \nu_{\mathscr{A}\parallel\mathscr{B}},\ t_{\mathscr{A}\parallel\mathscr{B}}) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}),\ \nu'_{\mathscr{A}\parallel\mathscr{B}},\ t'_{\mathscr{A}\parallel\mathscr{B}})$ exists on MLTS$((\mathscr{A},\tau) \parallel (\mathscr{B},\tau))$, with $\nu_{\mathscr{A}\parallel\mathscr{B}}$ and $\nu'_{\mathscr{A}\parallel\mathscr{B}}$ defined as $\nu_{\mathscr{A}\parallel\mathscr{B}}(x) = \nu_{\mathscr{A}}(x) \uplus \nu_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$, $\nu'_{\mathscr{A}\parallel\mathscr{B}} = \nu_{\mathscr{A}\parallel\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$:

(a) For $a \in \Sigma_{\mathscr{A}} \cap \Sigma_{\mathscr{B}}$: Let $((s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}),\ (s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}}, \nu'_{\mathscr{A}}, t'_{\mathscr{A}}),\ (s'_{\mathscr{B}},\ \nu'_{\mathscr{B}}, t'_{\mathscr{B}}))$ be a transition of MLTS$(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$. By Definition 78 (a), we have that the transition $((s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}},\ \nu'_{\mathscr{A}},\ t'_{\mathscr{A}}),\ (s'_{\mathscr{B}}, \nu'_{\mathscr{B}}, t'_{\mathscr{B}}))$ can not exist iff unless $(s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}) \xrightarrow{a}_{mlts} (s'_{\mathscr{A}}, \nu'_{\mathscr{A}},\ t'_{\mathscr{A}})$ in MLTS$(\mathscr{A},\tau)$ and $(s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}}) \xrightarrow{a}_{mlts} (s'_{\mathscr{B}}, \nu'_{\mathscr{B}}, t'_{\mathscr{B}})$ in MLTS $(\mathscr{B},\ \tau)$ both exist and the actions are synchronized. Therefore, since the two transitions on the MLTS, then we also know that there are two transitions on the corresponding MTA, $s_{\mathscr{A}} \xrightarrow{a,\phi_{\mathscr{A}},Y_{\mathscr{A}}}_{\textsf{ta}} s'_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{a,\phi_{\mathscr{B}},Y_{\mathscr{B}}}_{\textsf{ta}} s'_{\mathscr{B}}$. Additionally, we have the clock valuations $v_{\mathscr{A}}$ and $v'_{\mathscr{A}}$ (and $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$ respectively) of each state defined respectively as $v'_{\mathscr{A}} = v_{\mathscr{A}}[Y_{\mathscr{A}} \leftarrow 0]$ and $v_{\mathscr{A}} \models Inv(s'_{\mathscr{A}})$ and similarly in order for $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$. Hence, the composition of $s_{\mathscr{A}} \xrightarrow{a,\phi_{\mathscr{A}},Y_{\mathscr{A}}}_{\textsf{ta}} s'_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{a,\phi_{\mathscr{B}},Y_{\mathscr{B}}}_{\textsf{ta}} s'_{\mathscr{B}}$ at the MTA level is also based on the discrete transition of the MTA composition. This leads to the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{a}_{\textsf{ta}} (s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}})$. The clock valuation $v_{\mathscr{A}}$ (and $v_{\mathscr{B}}$) of each MTA $\mathscr{A}$ (and $\mathscr{B}$) is projected on the result of their composition. By Definition 78, we have that the result of the composition of MLTS there exists three kinds of transitions and the clock valuation $v_{\mathscr{A}}$ (and $v_{\mathscr{B}}$) is known for all kinds of transitions at the result of composition of MTA, then we can generalize this fact to every transition $\rightarrow_{mlts}$ on the corresponding MLTS. The clock $v'_{\mathscr{A}}$ (and $v'_{\mathscr{B}}$) is projected into the synchronization of the composition of the MTA. Then, the clock valuation that were reset by $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ will be reset by $v_{\mathscr{A}\|\mathscr{B}}$. Based on the composition of the MTA, we know that the discrete transition that are enabled by $(s_{\mathscr{A}}, s'_{\mathscr{A}})$ (and $(s_{\mathscr{B}}, s'_{\mathscr{B}})$) will be enabled by $(s_{\mathscr{A}}, s'_{\mathscr{A}}), (s_{\mathscr{B}}, s'_{\mathscr{B}})$. This leads to $v'_{\mathscr{A}\|\mathscr{B}} = v_{\mathscr{A}\|\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$. Therefore, given the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{a}_{\textsf{ta}} (s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}})$ and clock valuations $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$ then, we can obtain $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}), \xrightarrow{a}_{\textsf{mlts}} ((s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$.

Since the three cases hold, we conclude that $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\textsf{mlts}} ((q^{'}_{\mathscr{A}}, v^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, v^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\textsf{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\textsf{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ implies $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\textsf{mlts}} ((q^{'}_{\mathscr{A}}, v^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, v^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$.

(ii) Delay transition: Let $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$ be two states of MLTS$(\mathscr{A},\tau)$ and, MLTS$(\mathscr{B},\tau)$ respectively. A transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ exists on MLTS$(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$ iff the transition $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}}))$ exists on MLTS$((\mathscr{A},\tau) \parallel (\mathscr{B},\tau))$, with $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$ defined as $v_{\mathscr{A}\|\mathscr{B}}(x) = v_{\mathscr{A}}(x) \uplus v_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$, $v'_{\mathscr{A}\|\mathscr{B}} = v_{\mathscr{A}\|\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$:

(a) Let $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ be a transition of MLTS$(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$. By Definition 78 (ii), we have that the transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ can not exist iff unless $(s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}) \xrightarrow{\vec{d}}_{mlts} (s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}})$ in MLTS$(\mathscr{A},\tau)$ and $(s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}})$ in MLTS$(\mathscr{B},\tau)$ both exist. Therefore, since the two transitions on the MLTS, then we also know that there are two transitions on the corresponding MTA $q_{\mathscr{A}} \xrightarrow{\vec{d}}_{\textsf{ta}} q^{'}_{\mathscr{A}}$ and $q_{\mathscr{B}} \xrightarrow{\vec{d}}_{\textsf{ta}} q^{'}_{\mathscr{B}}$ where $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$. Additionally, we have the clock valuations $v_{\mathscr{A}}$ and $v'_{\mathscr{A}}$ (and $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$ respectively) of each state defined respectively as $\vec{d} = \tau(t'_{\mathscr{A}}) - \tau(t_{\mathscr{A}})$ and $\forall t \in [t_{\mathscr{A}}, t'_{\mathscr{A}}]$: $v +_{\pi} (\tau(t) - \tau(t_{\mathscr{A}})) \models Inv(s_{\mathscr{A}})$ and similarly in order for $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$. Hence, the composition of $s_{\mathscr{A}} \xrightarrow{\vec{d}}_{\text{icTA}} s'_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{\vec{d}}_{\text{ta}} s'_{\mathscr{B}}$ at the MTA level is also based on the delay transition of the MTA composition. This leads to the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{ta}} (s'_{\mathscr{A}}, s'_{\mathscr{B}})$. The clock valuation $v_{\mathscr{A}\|\mathscr{B}}$ of each MTA is projected on the result of their composition. Then, the clock valuation that were reset by $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ will be reset by $v_{\mathscr{A}\|\mathscr{B}}$. Based on the composition of the MTA, we know that the delay transition that are enabled by $(s_{\mathscr{A}}, s'_{\mathscr{A}})$ and $(s_{\mathscr{B}}, s'_{\mathscr{B}})$ will be enabled by $(s_{\mathscr{A}}, s_{\mathscr{B}}), (s'_{\mathscr{A}}, s'_{\mathscr{B}})$. Therefore, given the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{ta}} (s'_{\mathscr{A}}, s'_{\mathscr{B}})$ and clock valuations $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$ then, we can obtain $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}), \xrightarrow{\vec{d}}_{\text{mlts}} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$.

(b) Let $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}}))$ be a transition of MLTS$(\mathscr{A} \parallel \mathscr{B}, \tau)$. By Definition 78 (ii), we have that the transition $(q_{\mathscr{A}}, q_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} (q'_{\mathscr{A}}, q'_{\mathscr{B}})$ and the clock valuation $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$. Therefore, since the two transitions on the composition of $\mathscr{A}$ and $\mathscr{B}$, then by the discrete transition of the composition of the two MTA, we know that $q_{\mathscr{A}} \xrightarrow{\vec{d}}_{\text{ta}} q'_{\mathscr{A}}$ and $q_{\mathscr{B}} \xrightarrow{\vec{d}}_{\text{ta}} q'_{\mathscr{B}}$ both transition exist and $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$. Based on the composition of the clock valuation $v'_{\mathscr{A}\|\mathscr{B}}$ is defined respectively as $\vec{d} = \tau(t'_{\mathscr{A}\|\mathscr{B}}) - \tau(t_{\mathscr{A}\|\mathscr{B}})$ and $\forall t \in [t_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}}] : v_{\mathscr{A}\|\mathscr{B}} +_{\pi} (\tau(t) - \tau(t_{\mathscr{A}\|\mathscr{B}})) \models Inv_{\mathscr{A}}(s_{\mathscr{A}}) \wedge Inv_{\mathscr{B}}(s_{\mathscr{B}})$ and similarly in order for $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$. Hence, the two transitions at the MTA level along with the clock valuation of $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$, is applied which leads into $(s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}) \xrightarrow{\vec{d}}_{\text{ta}} (s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}})$ of MLTS$((\mathscr{A}, \tau)$ and $(s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{ta}} (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}})$ of MLTS$((\mathscr{B}, \tau)$. Now, the composition of these two transitions at the MLTS level is also based on the of discrete transition the MTA composition. This leads into the transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ which happens to be our awaited conclusion.

Since the two implications hold, we conclude that $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((q'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (q'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ implies $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((q'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (q'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$.

Since the two implications hold, we conclude that $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\text{mlts}} ((q'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (q'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\text{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\text{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$

implies $((q_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\mathsf{mlts}} ((q^{'}_{\mathscr{A}}, \nu^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, \nu^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$.

Since (1), (2) and (3), we conclude that for any $\tau \in Rates$, $\mathsf{MLTS}(\mathscr{A}, \tau) \parallel \mathsf{MLTS}(\mathscr{B}, \tau) \approx \mathsf{MLTS}(\mathscr{A} \parallel \mathscr{B}, \tau)$. □

proposition 12. *Let $\mathscr{M}_1$ and $\mathscr{M}_2$ be two MLTS over the actions Σ, then $\mathscr{M}_1 \parallel \mathscr{M}_2 \approx \mathscr{M}_2 \parallel \mathscr{M}_1$.*

Proof. The proof of this proposition consists in showing that each transition of $\mathscr{M}_1 \parallel \mathscr{M}_2$ can be found in $\mathscr{M}_2 \parallel \mathscr{M}_1$ and vice versa, where $\mathscr{R}$ obey the symmetric property, i.e., $(q_{\mathscr{M}_1}, q_{\mathscr{M}_2})\ \mathscr{R}\ (q_{\mathscr{M}_2}, q_{\mathscr{M}_1})$. Based on the MLTS composition, there exists two types of transitions on the resulting system. Let $\mathscr{R} = \{(q_{\mathscr{M}_1}, q_{\mathscr{M}_2}) | (q_{\mathscr{M}_2}, q_{\mathscr{M}_1}) \in Q_{\mathscr{M}_2 \parallel \mathscr{M}_1}\}$. It directly follows from the definition of parallel composition in MLTS (Definition 24) that:

(i) For any discrete transition $(q_{\mathscr{M}_1}, q_{\mathscr{M}_2}) \xrightarrow{a}_{\mathscr{M}_1 \parallel \mathscr{M}_2} (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})$ with $a \in \Sigma$, there exists a corresponding transition $(q_{\mathscr{M}_2}, q_{\mathscr{M}_1}) \xrightarrow{a}_{\mathscr{M}_2 \parallel \mathscr{M}_1} (q'_{\mathscr{M}_2}, q'_{\mathscr{M}_1})$ with $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), (q_{\mathscr{M}_2}, q_{\mathscr{M}_1})) \in \mathscr{R}$.

(ii) For any delay transition $(q_{\mathscr{M}_1}, q_{\mathscr{M}_2}) \xrightarrow{d}_{\mathscr{M}_1 \parallel \mathscr{M}_2} (q'_{\mathscr{M}_1}, q'_{\mathscr{M}_2})$ with $d \in \mathbb{R}^n_{\geq 0}$, there exists a corresponding transition $(q_{\mathscr{M}_2}, q_{\mathscr{M}_1}) \xrightarrow{d}_{\mathscr{M}_2 \parallel \mathscr{M}_1} (q'_{\mathscr{M}_2}, q'_{\mathscr{M}_1})$ with $((q_{\mathscr{M}_1}, q_{\mathscr{M}_2}), (q_{\mathscr{M}_2}, q_{\mathscr{M}_1})) \in \mathscr{R}$.

Since every initial state $(q^0_{\mathscr{M}_1}, q^0_{\mathscr{M}_2})$ of $\mathscr{M}_1 \parallel \mathscr{M}_2$ has a match $(q^0_{\mathscr{M}_2}, q^0_{\mathscr{M}_1})$ in the initial states of $\mathscr{M}_2 \parallel \mathscr{M}_1$, and $((q^0_{\mathscr{M}_1}, q^0_{\mathscr{M}_2}), (q^0_{\mathscr{M}_2}, q^0_{\mathscr{M}_1})) \in \mathscr{R}$. Therefore, $\mathscr{R}$ is a bisimulation for $\mathscr{M}_1 \parallel \mathscr{M}_2 \approx \mathscr{M}_2 \parallel \mathscr{M}_1$. Finally, by following a similar step, we could show that $\mathscr{M}_2 \parallel \mathscr{M}_1 \approx \mathscr{M}_1 \parallel \mathscr{M}_2$. □

In the context of multi-timed bisimulation, compositionality [BCG88] is captured by the following definition:

Definition 82 (Compositionality). *A binary relation $\approx$ between two MLTS $\mathscr{M}_1, \mathscr{M}_2$ is compositional if $\mathscr{M}_1 \approx \mathscr{M}_2$ and $\mathscr{M}_3 \approx \mathscr{M}_4$ implies $\mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_4$.*

proposition 13. *Let $\mathscr{M}_1$, $\mathscr{M}_2$ and $\mathscr{M}_3$ be three MLTS over the set of actions Σ. For any $\mathscr{M}_1$, $\mathscr{M}_2$ and $\mathscr{M}_3$, $\approx$ is compositional if and only if $\mathscr{M}_1 \approx \mathscr{M}_2 \Rightarrow \mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_3$ (invariant under composition).*

Proof. The proof of this proposition consists in showing for the sufficient direction that $\mathscr{M}_1 \approx \mathscr{M}_2 \Rightarrow \mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_3$. Assume that $\approx$ is compositional. Since $\approx$ is reflexive $\mathscr{M}_3 \approx \mathscr{M}_3$ holds. Using the definition of compositionality, $\mathscr{M}_1 \approx \mathscr{M}_2$ and $\mathscr{M}_3 \approx \mathscr{M}_3$ imply $\mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_3$. For the necessary direction, assume that $\approx$ is invariant under composition. Then $\mathscr{M}_1 \approx \mathscr{M}_2$ implies $\mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_3$. Also, $\mathscr{M}_3 \approx \mathscr{M}_4$ and commutativity of composition implies $\mathscr{M}_2 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_4$, and by transitivity $\mathscr{M}_1 \parallel \mathscr{M}_3 \approx \mathscr{M}_2 \parallel \mathscr{M}_4$. □

7.4 Decidability of Multi-timed Bisimulation

A fundamental decision problem for TA arises from the comparison of two automata that can be structurally equivalent (i.e., have the same locations and transitions). In TA it was shown that timed trace inclusion is undecidable [AD94], whereas timed bisimulation is decidable, making timed bisimilarity a particularly useful equivalence notion for verifying RTS. Thus, methods based on symbolic abstractions such as the region-based [AD94] and the zone-based abstraction [BY04] are used for verifying timed bisimulation. However, the region-based method constructs a finite region graph, but the problem with this graph is generally the potential explosion in the number of regions (i.e., $\mathcal{O}(|X|! \cdot c^{|X|})$ where X is a set of clocks and c are the maximal constants appearing in the clock constraints). Instead, zone-based methods construct a zone graph, which is essentially a more efficient representation of the state space for TA [BY04]. Zones can be represented and manipulated efficiently, but abstractions of zones are required for checking timed bisimilarity [WL97] [GMS18].

As in TA, we present here a fundamental decidable problem used to reason about behavioral equivalence between different components of a DRTS. Our strong multi-timed bisimulation can be used to reason about the complete computational steps of two MTA. Verification of strong multi-timed bisimulation is a challenging problem, because the state space explosion caused by both interactions between components and (independent) local clocks must be taken into account. Therefore, we use the zone-based abstraction here, and inspired by [LLW95], we show that our multi-timed bisimulation is decidable over an appropriate multi-timed zone graph.

7.4.1 Multi-Clock Zones

Here, we define a symbolic state (or zone) as a pair $q = (s, \mathcal{Z})$, where s is a location of the MTA $\mathscr{A}$ and $\mathcal{Z}$ is a clock zone. A symbolic state $q = (s, \mathcal{Z})$ represents all the states $(s', v) \in q$ if $s = s'$ and $v \in \mathcal{Z}$, indicating that a state is contained in a zone. Similarly, we can write $(s, \mathcal{Z}) \subseteq (s', \mathcal{Z}')$ to indicate that $s = s'$ and $\mathcal{Z} \subseteq \mathcal{Z}'$. To define the notion of a clock zone [LLW95] over a set of clocks X, we will consider the set $\Phi^+(X)$ of all the diagonal constraints over X. Formally, a clock zone is described by a conjunction of diagonal constraints, i.e., constraints made up of a conjunction of inequalities that compare either a clock value or the difference between two clock values to an integer. However, preserving the difference between clocks makes sense in the setting of TA, because as time passes, the clock differences remain the same (i.e., the clock difference is an invariant with time), since all clocks evolve at the same rate. But because our clocks evolve at rates that can be independent of each other, we need to adapt certain operations on clock zones. This is described by rate constraints.

Operations on Multi-Clock Zones Here we extend the semantics of some operations on clock zones to their multi-timed version (time successor and predecessor). The operations intersection, limiting projection, clock reset, and inverse clock reset retain the same semantics as in section 5.3.4.1. Mote that clock zones are always convex. To implement our decidable algorithms, we need to be able to compute successors and predecessors of zones for delay and action transitions of MTA.

Definition 83 (**Operations on Multi-clock zones**). *Let $\mathcal{Z}$ be a clock zones. The semantics of the time successor and time predecessor on a clock zone can be defined:*

(i) Time successor: $\mathcal{Z}\uparrow = \{v + \vec{d} \mid v \in \mathcal{Z},\ \vec{d} \in \mathbb{R}_{\geq 0}^{Proc}, \exists t, t' > 0, t \leq t',\ and\ \exists \tau \in Rates, \vec{d} = \tau(t') - \tau(t)\}$,

(ii) Time predecessor: $\mathcal{Z}\downarrow = \{v - \vec{d} \mid v \in \mathcal{Z},\ \vec{d} \in \mathbb{R}_{\geq 0}^{Proc}, \exists t, t' > 0, t \leq t',\ and\ \exists \tau \in Rates, \vec{d} = \tau(t') - \tau(t)\}$.

proposition 14. *Let $\mathcal{Z}, \mathcal{Z}'$ be two clock zones and $Y \subseteq X$. Then $\mathcal{Z} \cap \mathcal{Z}'$, $\mathcal{Z}\rfloor_Y$, $\mathcal{Z}\downarrow_Y$, $\mathcal{Z}\uparrow_Y$, $\mathcal{Z}\uparrow$, and $\mathcal{Z}\downarrow$ are also clock zones.*

Proof. Let $\mathcal{Z}, \mathcal{Z}'$ be zones and $Y \subseteq X$, then we need to prove the following operations are clock zones:

(i) $\mathcal{Z} \cap \mathcal{Z}'$,
(ii) $\mathcal{Z}\rfloor_Y$,
(iii) $\mathcal{Z}\downarrow_Y = \mathcal{Z}\rfloor_{X\setminus Y} \cap (\bigwedge_{y\in Y} y = 0)$,
(iv) $\mathcal{Z}\uparrow_Y = (\mathcal{Z} \cap (\bigwedge_{y\in Y} y = 0))\rfloor_{X\setminus Y}$,
(v) $\mathcal{Z}\uparrow$,
(vi) $\mathcal{Z}\downarrow$,

(i) $\mathcal{Z} \cap \mathcal{Z}'$ is a conjunction of clock constraints and, therefore, a clock zone.

(ii) Let $\mathcal{Z}$ be a clock zone and $Y \subseteq X$ be a set of clocks. We are going to prove the second case (ii) by showing that every clock valuation that satisfies $\mathcal{Z}\rfloor_Y$ also satisfies $\mathcal{Z}_\phi$ where $\phi \in \Phi(X)^+$. For a clock constraint $\phi \in \Phi(X)^+$, let $\phi\rfloor_Y$ be the constraint, where all propositions containing clocks of the set $X\setminus Y$ are removed. Furthermore, for a constraint $\phi \in \Phi(Y)$, let $\phi\rfloor_X$ be the constraint, where all propositions containing clocks in $X\setminus Y$. Thus, for the clock zone $\mathcal{Z}$ $= \{v \mid v \models \phi\}$ of a constraint, $\phi \in \Phi(X)^+$ the clock zone projection operation can be defined as $\mathcal{Z}\rfloor_Y = \{v \mid v \models \phi\rfloor_Y\}$ of a constraint $\phi \in \Phi(X)^+$. Thus, $\mathcal{Z}_\phi \subseteq \mathcal{Z}_\phi\rfloor_Y$.

(iii) Let $\mathcal{Z}$ be a clock zone and $Y \subseteq X$ be a set of clocks. We are going to prove the third case (iii) by showing that every clock valuation that is in $\mathcal{Z}\downarrow_Y$ is also in $\mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$ and vice versa (i.e., $\mathcal{Z}\downarrow_Y \subseteq \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$ and $\mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0) \subseteq \mathcal{Z}\downarrow_Y$).

(a) $\mathcal{Z}\downarrow_Y \subseteq \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$: Consider an arbitrary clock valuation v. Then, we assume $v \in \mathcal{Z}\downarrow_Y$ and show $v \in \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$. Since v $\in \mathcal{Z}\downarrow_Y$ and clock valuation v', $\mathcal{Z}\downarrow_Y$ denotes the valuation v' such that for all $y \in Y$, $v(y) = 0$ and for all $x \in X \setminus Y$, $v'(x) = v(x)$ (by Definition 83 (3)). Hence, we can deduce from the above proposition and Definition 83 (2) that if $v \in \mathcal{Z}\downarrow_Y$, then $v \in (\bigwedge_{y\in Y} v(y) = 0)$ and $v \in \mathcal{Z}\rfloor_Y$. Therefore, by definition of intersection of zone, $v \in (\bigwedge_{y\in Y} v(y) = 0) \cap \mathcal{Z}\rfloor_Y$. Thus, we have $\mathcal{Z}\downarrow_Y \subseteq (\bigwedge_{y\in Y} v(y) = 0) \cap \mathcal{Z}\rfloor_Y$ are also clock zone.

(b) $\mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0) \subseteq \mathcal{Z}\downarrow_Y$: Consider an arbitrary clock valuation v. Then, we assume $v \in \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$ and show $v \in \mathcal{Z}\downarrow_Y$. By intersection of zones, we know $v \in \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$ implies both $v \in \mathcal{Z}\rfloor_Y$ and $v \in (\bigwedge_{y\in Y} v(y) = 0)$. Therefore, for a subset of clocks $Y \subseteq X$ and clock valuation v', $\mathcal{Z}\rfloor_Y$ denotes the valuation v' such that for all $x \in X \setminus Y$, $v'(x) = v(x)$ and $(\bigwedge_{y\in Y} v(y) = 0)$ denotes the valuation v such that for all $y \in Y$ $v(y) = 0$. This is exactly the definition of clock reset, and so $v \in \mathcal{Z}\downarrow_Y$. Thus, we have $\mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0) \subseteq \mathcal{Z}\downarrow_Y$ are also clock zone.

The proof of these zones establishes the desired equality $\mathcal{Z}\downarrow_Y = \mathcal{Z}\rfloor_Y \cap (\bigwedge_{y\in Y} v(y) = 0)$.

(iv) $\mathcal{Z}\uparrow_Y = (\mathcal{Z} \cap (\bigwedge_{y\in Y} v(y) = 0))\rfloor_Y$. The argument is symmetric to (iii).

(v) Let $\mathcal{Z}$ be a clock zone and X be a set of clocks. We are going to demonstrate the fifth case (v) by showing that if $\mathcal{Z}$ is a clock zone, then a clock valuation $v' \in \mathcal{Z}\uparrow$, if there exists a tuple $\vec{d} \in \mathbb{R}^X_{\geq 0}$, a tuple $\tau \in Rates$, $t, t' \geq 0$, $t \leq t'$ with $\vec{d} = \tau(t') - \tau(t)$, there exists a clock valuation $v \in \mathcal{Z}$ such that $v + \vec{d} = v'$. Thus, $\mathcal{Z}\uparrow$ is a clock zone. In order to demonstrate this, we need to solve the following system of inequalities:

$$\begin{cases} -c_{0,i} - v(x_i) \sim \vec{d} & for\ all\ x_i \in X \\ \vec{d} \sim c_{i,0} - v(x_i) & for\ all\ x_i \in X \\ v(x_i + \vec{d}) - v(x_j + \vec{d}) \sim c_{i,j} & for\ all\ x_i, x_j \in X \\ \vec{d} \geq 0 \end{cases}$$

(a) Let x_i be an independent clock, such that $x_i \in X$. For all $v \in \mathcal{Z}$, we know that if $-c_{0,i}$ - $v(x_i) \sim \vec{d}$ then $-c_{0,i} \sim v'(x_i)$ from which we can deduce that the inequality does not force $\vec{d} \in \mathbb{R}^X_{\geq 0}$ to be negative. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). Let $\vec{d} \in \mathbb{R}^X_{\geq 0}$ be such solution. We let v be the valuation such that $v(x_i) = v'(x_i) - \vec{d}$ for all $x_i \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $v'(x_i) = v(x_i) + \vec{d}$ with $v \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^X_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $v' \in \mathcal{Z}\uparrow$ and $\mathcal{Z}\uparrow$ is a clock zone.

(b) Let x_i be an independent clock such that $x_i \in X$. For all $v \in \mathcal{Z}$, we know that if $\vec{d} \sim c_{i,0} - v(x_i)$ then $v'(x_i) \sim c_{i,0}$ from which we can deduce that the inequality does not force $\vec{d} \in \mathbb{R}^X_{\geq 0}$ to be negative. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). Let $\vec{d} \in \mathbb{R}^X_{\geq 0}$ be such solution. We let v be the valuation such that $v(x_i) = v'(x_i) - \vec{d}$ for all $x_i \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $v'(x_i) = v(x_i) + \vec{d}$ with $v \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^X_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $v' \in \mathcal{Z}\uparrow$ and $\mathcal{Z}\uparrow$ is a clock zone.

(c) Let x_i, x_j be two independent clocks such that x_i, $x_j \in X$. For all $v \in \mathcal{Z}$, we know that $(v(x_i) + \vec{d})$ - $(v(x_j) + \vec{d}) \sim c_{i,j}$, but, due to the fact that the clocks evolve at rates that can be independent of each other, clock differences can change over time and the two occurrences of $\vec{d}$

do not cancel each other out, then we can deduce that the inequality $\nu'(x_i)$ - $\nu'(x_j) \sim c_{i,j}$ is already in the appropriate form. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). Let $\vec{d} \in \mathbb{R}^X_{\geq 0}$ be such solution. We let ν be the valuation such that $\nu(x_i) = \nu'(x_i) - \vec{d}$ and $\nu(x_j) = \nu'(x_j) - \vec{d}$ for all x_i, $x_j \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $\nu'(x_i) = \nu(x_i) + \vec{d}$ and $\nu'(x_j) = \nu(x_j) + \vec{d}$ with $\nu \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^X_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $\nu' \in \mathcal{Z} \uparrow$ and $\mathcal{Z} \uparrow$ is a clock zone.

(vi) $\mathcal{Z} \downarrow$. The argument is symmetric to (v).

□

7.4.2 Multi-timed Zone Graphs

As in TA, MTA cannot be analyzed by finite state techniques, since the MLTS associated with it has infinitely many states. Therefore, it must be analyzed symbolically. Here, we define the multi-timed zone graph using the independent local clocks, and we extend the well-known zone graph for TA [AD94] [LLW95]. We work with the multi-timed zone graph as a symbolic representation. The elements of a zone graph are symbolic states (i.e., $q = (s, \mathcal{Z})$). We will use the notation $\mathsf{Action}(e)$ to denote the action a of an edge e. The initial state $q_0 = (s_0, \nu_0)$ in $\mathcal{A}$ corresponds to a symbolic state $(s_0, \mathcal{Z}_0)$. The backward or forward exploration through the state space of MTA requires operations on clock zones to return the discrete predecessors or discrete successors of a symbolic state. More precisely, we extend the symbolic discrete successor and predecessor operations on clock zones for a MTA $\mathcal{A}$. These operations on clock zones can be implemented efficiently on DBM [Dil90] [BY04]. We can now define the symbolic discrete successor and predecessor operations on clock zones as follows:

Definition 84 (**Discrete Successor**). *Let $q = (s, \mathcal{Z})$ be a zone and e=$(s, a, \phi, Y, s') \in \rightarrow_{ta}$ be a transition of $\mathcal{A}$, then post$(\mathcal{Z}, e)$=$\{\nu' \mid \exists \nu \in \mathcal{Z}, \exists \tau \in Rates, (s, \nu) \xrightarrow{e}_{mlts(\mathcal{A},\tau)} (s', \nu')\}$ is the set of valuations that q can reach by taking the transition e.*

Intuitively, the zone $(s', \mathsf{post}(\mathcal{Z}, e))$ describes the discrete successor of the zone $(s, \mathcal{Z})$ under the transition e.

Definition 85 (**Discrete Predecessor**). *Let $q = (s', \mathcal{Z}')$ be a zone and e=$(s, a, \phi, Y, s') \in \rightarrow_{ta}$ be a transition of $\mathcal{A}$, then pred$(\mathcal{Z}', e)$=$\{\nu \mid \exists \nu' \in \mathcal{Z}', \exists \tau \in Rates, (s, \nu) \xrightarrow{e}_{mlts(\mathcal{A},\tau)} (s', \nu')\}$ is the set of valuations that q can reach by executing the transition e.*

The zone $(s, \mathsf{pred}(\mathcal{Z}', e))$ describes the discrete predecessor of the zone $(s', \mathcal{Z}')$ under the transition e. The set $\mathsf{pred}(\mathcal{Z}', e)$ can be computed using the operations inverse clock reset and intersection on clock zones as follows:

$$\mathsf{pred}(\mathcal{Z}', e) = ((\mathcal{Z}' \uparrow_Y \cap \phi) \cap Inv(s)).$$

The set $\mathsf{post}(\mathcal{Z}, e)$ can be obtained using the operations clock reset and the standard intersection of clock zones as follows:

$$\mathsf{post}(\mathcal{Z}, e) = ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s')).$$

The sets $\mathsf{post}(\mathcal{Z}, e)$ and $\mathsf{pred}(\mathcal{Z}', e)$ are thus also clock zones.

proposition 15. *Let $\mathcal{A}$ be a MTA, $e = (s, a, \phi, Y, s') \in \rightarrow_{mta}$ be a transition of a MTA $\mathcal{A}$ and $(s, \mathcal{Z})$ be a zone, then post$(\mathcal{Z}, e) = ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_X \cap Inv(s'))$ and pred$(\mathcal{Z}', e) = ((\mathcal{Z}' \uparrow_X \cap \phi) \cap Inv(s))$.*

Proof. Let $(s, \mathcal{Z})$ be a zone and $e = (s, a, \phi, Y, s') \in \rightarrow_{mta}$ be a transition of an MTA $\mathcal{A}$, then we need to prove the following equalities:

(i) $\mathsf{post}(\mathcal{Z}, e) = ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$,

(ii) $\mathsf{pred}(\mathcal{Z}', e) = ((\mathcal{Z}' \uparrow_X \cap \phi) \cap Inv(s))$.

(i) Let $\mathcal{Z}$ be a convex clock zone and X be a set of clocks. We are going to prove the third case (i) by showing that every clock valuation that is in $\mathsf{post}(\mathcal{Z}, e)$ is also in $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$ and vice versa (i.e., $\mathsf{post}(\mathcal{Z}, e) \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$ and $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s')) \subseteq \mathsf{post}(\mathcal{Z}, e)$.

(a) $\mathsf{post}(\mathcal{Z}, e) \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$: Consider an arbitrary clock valuation v. Then, we assume $v \in \mathsf{post}(\mathcal{Z}, e)$ and show $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$. Since $v \in \mathsf{post}(\mathcal{Z}, e)$, then, $v \in \mathcal{Z}$ and exists $t \in \mathbb{R}_{\geq 0}$ and v' such that $(s, v, t) \xrightarrow{e}_{mlts} (s', v', t)$ where e is a transition. By Definition 75 (2)) there exists a discrete transition between s and s' with $v \in \phi$ and $v' = v[X \leftarrow 0]$ and $v' \in Inv(s')$. By Definition 83(3) it follows that $v \in \mathcal{Z}$. Therefore, by intersection of zones and Definition 83(4), we have that $v \in (\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X$ and $v \in Inv(s')$ then by clock equivalence ($v \equiv v'$) (Definition 41) $v \in (\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap v \in Inv(s')$. Thus, we have $\mathsf{post}(\mathcal{Z}, e) \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$ are also zone.

(b) $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s')) \subseteq \mathsf{post}(\mathcal{Z}, e)$: Consider an arbitrary clock valuation v. Then, we assume $v \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$ and show $v \in \mathsf{post}(\mathcal{Z}, e)$. Since $v \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$, then, $v \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y$ and $v \in Inv(s')$. By Definition 83(3) and conjunction of zones, we have that $v \in ((\mathcal{Z} \cap (\phi \cap Inv(s))$ and $v \in Inv(s')$, then $v \in (\mathcal{Z} \cap (\phi \cap Inv(s))) \cap Inv(s') \subseteq \mathsf{post}(\mathcal{Z}, e)$. This is exactly the definition of inclusion of zones $v \in \mathsf{post}(\mathcal{Z}, e)$. Thus, we have $\mathsf{post}(\mathcal{Z}, e) \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$ are also zone.

The proof of these zones establishes the desired equality $\mathsf{post}(\mathcal{Z}, e) = ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$.

(ii) $\mathsf{pred}(\mathcal{Z}', e) = ((\mathcal{Z}' \uparrow_Y \cap \phi) \cap Inv(s))$. The argument is symmetric to (ii).

□

A symbolic semantics of MTA called multi-timed zone graph is defined as follows:

Definition 86 (**Multi-timed Zone Graph**). *Given a MTA $\mathcal{A}$, its infinite multi-timed zone graph (MZG($\mathcal{A}$)) is a transition system MZG($\mathcal{A}$) $= (Q, q_0, (\Sigma \cup \{\epsilon\}), \rightarrow_{MZG})$, where:*

(i) Q consists of pairs $q = (s, \mathcal{Z})$ where $s \in S$, and $\mathcal{Z} \in \Phi^+(X)$ is a non-empty clock zone with $\mathcal{Z} \subseteq Inv(s)$,

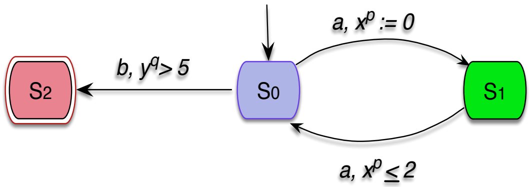


Figure 7.6: A MTA $\mathscr{A}$

(ii) *$q_0 \in Q$ is the initial zone $q_0 = (s_0, \mathcal{Z}_0)$ with $\mathcal{Z}_0 = [\![\bigwedge_{x \in X} x = 0]\!]$,*

(iii) *Σ is the set of labels of $\mathscr{A}$,*

(iv) *$\rightarrow_{MZG} \subseteq Q \times (\rightarrow_{ta} \cup \{\epsilon\}) \times Q$ is a set of transitions, where each transition in MZG($\mathscr{A}$) is a labelled by a transition $e = (s, a, \phi, Y, s') \in \rightarrow_{ta}$, where s and s' are the source and target locations, ϕ is a clock constraint defining the guard of the transition, a is the action of the edge and Y is the set of clocks to be reset by the transition in the MTA $\mathscr{A}$. For each $e \in \Sigma$, transitions are defined by the rules:*

  (a) *For every $e = (s, a, \phi, Y, s')$ in $\mathscr{A}$ and zone $(s, \mathcal{Z})$ already in Q, there exists a discrete transition (q, e, q'), where $q = (s, \mathcal{Z}) \xrightarrow{e}_{MZG} q' = (s', post(\mathcal{Z}, e))$ if $post(\mathcal{Z}, e) \neq \emptyset$.*

  (b) *For a clock zone $\mathcal{Z}$, there exists a delay transition (q, ϵ, q'), where $q = (s, \mathcal{Z}) \xrightarrow{\epsilon}_{MZG} q' = (s, \mathcal{Z}')$ and $\mathcal{Z}' = \mathcal{Z} \uparrow \cap\, Inv(s)$ where $\mathcal{Z}'$ is called a time successor of zone $\mathcal{Z}$.*

Every zone has a ϵ delay transition to itself, and the ϵ transitions are also transitive. Furthermore, (1) a ϵ delay transition must be strict to have clock zones and, (2) it is not reflexive between zones. Since a ϵ transition is reflexive, time successor is also a reflexive relation. For both e and ϵ transitions, if $\mathcal{Z}$ is a zone then $\mathcal{Z}'$ is also a zone.

Example 25. *Consider the MTA $\mathscr{A}$ in Figure 7.6 with the finite input alphabet $\Sigma = \{a, b\}$, the set of processes $Proc = \{p, q\}$, the set of local clocks $X = \{x^p, y^q\}$. Figure 7.7 shows MZG($\mathscr{A}$).*

7.4.3 Deciding Reachability in Multi-timed Zone Graphs

The multi-timed zone graph can be infinite because the constants used in the zones can grow forever. However, a symbolic multi-timed zone graph can be made finite using the extrapolation technique (see Section 5.3.7 $Extra^+_{LU}$). Multi-timed zone graphs can be used as the basis for a reachability checking algorithm (see Section 5.3.4). The main idea of the algorithm is a depth-first search on the zone graph. The algorithm 7.1 formalizes this process to compute the reachability zone graph for a state q_0. The algorithm constructs a finite symbolic zone graph ($\mathsf{MZG}_{Extra^+_{LU}}(\mathscr{A})$), given a MTA ($\mathscr{A}$). However, since the multi-timed bisimilarity algorithm used here uses two MTA ($\mathscr{A}$ and $\mathscr{B}$), we must first construct a finite symbolic zone graph ($\mathsf{MZG}_{Extra^+_{LU}}(\mathscr{C})$), given the parallel composition of two MTA ($\mathscr{A}$ and $\mathscr{B}$) with the same actions $\Sigma_{\mathscr{A}} = \Sigma_{\mathscr{B}}$, but disjoint clocks $X_{\mathscr{A}} \cap X_{\mathscr{B}} = \emptyset$).

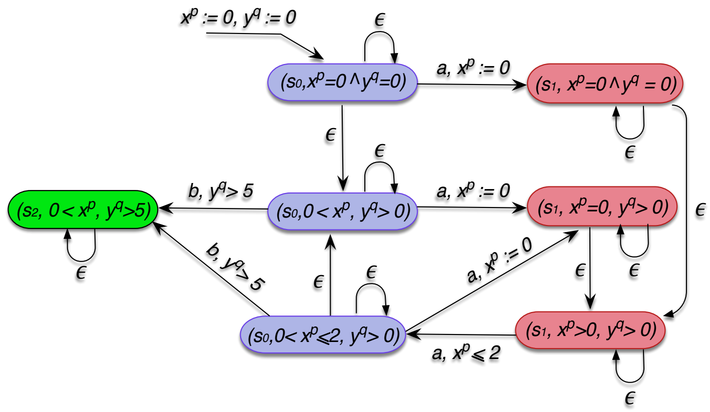


Figure 7.7: The zone graph for the automaton $\mathscr{A}$ in Figure 7.6

The Algorithm 7.1 is a classical algorithm (including subsumption) [BBLP06] except that the delay successors are adapted. Algorithm 7.1 builds a symbolic zone graph, starting with the pair $q_0 = (s_0, Extra^+_{LU}(\mathcal{Z}_0))$ where s_0 is the initial location of the automaton $\mathscr{A}$, $\mathcal{Z}_0 \leftarrow [\![\bigwedge_{x \in X} x = 0]\!]$ represents the initial zone and $Extra^+_{LU}$ (*LU*-bound) [Bou04a] [BBLP06] (line 9) is the extrapolation abstraction technique. For each location s of a $\mathsf{MZG}_{Extra^+_{LU}}(\mathscr{A})$, there are bounded functions L, U (see Section 5.3.4) and we can construct the symbolic zone graph by adding symbolic states of the form $q_{\mathsf{MZG}} = (s, Extra^+_{LU}(\mathsf{post}(\mathcal{Z}, e)))$. Algorithm 7.1 presents a forward reachability algorithm by using a waiting set of pairs D and a set of visited pairs Q such that D ⊆ Q and a set of transitions T_{ZG} in line 9. The algorithm consists of a loop that iterates over D in line 10. Initially, the set contains only the element $q_0 = (s_0, Extra^+_{LU}(\mathcal{Z}_0))$ in order to start the search at the starting location s_0. At each iteration, the algorithm takes a pair $\mathcal{Z}_1$ from D and removes it from D in line 11. The algorithm enters an inner loop (lines 13-24) and for each discrete transition $e = (s, a, \phi, Y, s')$ with $\mathcal{Z} \wedge \phi \neq \emptyset$ (i.e., applicable) the algorithm computes the successors of s and associates each successor with the zones $(s', Extra^+_{LU}(\mathsf{post}(\mathcal{Z}, e))) = \mathcal{Z}_2$ (lines 14-15). This result is a set of new pairs $(s, \mathcal{Z}) \in S \times \Phi^+(X)$. The transitions e with successors are stored in the set of labels T_{ZG} in line 15. In line 16, it is checked if there exists an already visited pair $(s', \mathcal{Z}_3) \in$ Q such that $\mathcal{Z}_2) \subseteq \mathcal{Z}_3$. If true, the discrete transition between $(s, \mathcal{Z}_1)$ $\xrightarrow{e}_{\mathsf{ZG}}$ $(s', \mathcal{Z}_3)$ is stored in the set of transitions T_{ZG} (line 17). Otherwise, the discrete transition between $(s, \mathcal{Z}_1) \xrightarrow{e}_{\mathsf{ZG}} (s', \mathcal{Z}_2)$ is stored in the set of transitions T_{ZG} and the successor pair $(s', \mathcal{Z}_2)$ is stored in Q and D (lines 20-22). During the search, the algorithm also computes the delay transition by the conjunction between the time successors of the current construction zone $\mathcal{Z}_1$ $(s, Extra^+_{LU}(\mathcal{Z}_1)\uparrow)$, the invariant constraint and rate constraint of the current location s (i.e., $Extra^+_{LU}(\mathcal{Z}_1 \uparrow \wedge Inv(s))$) in line 25. In line 26, it is checked if there exists an already visited pair $(s, \mathcal{Z}_3) \in$ Q such that $\mathcal{Z}_2 \subseteq \mathcal{Z}_3$. If true, the delay transition between $(s, \mathcal{Z}_1) \xrightarrow{e}_{\mathsf{ZG}} (s, \mathcal{Z}_3)$ is stored in the set of transitions T_{ZG} (line 28). Otherwise, the delay transition between $(s, \mathcal{Z}_1)$ $\xrightarrow{e}_{\mathsf{ZG}}$ $(s, \mathcal{Z}_2)$ is stored in the set of transitions T_{ZG} and the successor pair $(s, \mathcal{Z}_2)$ is

```
1  Input: A MTA $\mathscr{C}$=($S$,$s_0$,$\Sigma$,$X$,$\rightarrow_{ta}$,$Inv$,$F$)
2  Output: A reachable zone graph MZG($\mathscr{C}$) = $(Q, q_0, (\Sigma \cup \{\epsilon\}), T_{ZG})$
3  //$s \in S$ is a location of $\mathscr{C}$, $\mathcal{Z}_{1 \le i \le 3}$ are DBM
4  //$T_{ZG}$ is a set of transitions (i.e.  $\rightarrow_{ZG}$= $T_{ZG}$)
5  //D and Q are sets of pairs in $S \times \Phi^+(X)$
6  //D is the set of open states
7  MZG BuildSymbZoneGraph(MTA $\mathscr{C}$){
8    $q_0$ = $(s_0, Extra^+_{LU}(\mathcal{Z}_0))$ s.t for all $x$ ∈ $X$ and $v$ ∈ $\mathcal{Z}_0$, $v(x) = 0$;
9    Q, D = $\{q_0\}$, $T_{ZG}$ = $\emptyset$;
10    while{D != $\emptyset$}{
11     Choose and Remove $(s, \mathcal{Z}_1)$ from D;
12      for(transition $e$ = $(s, a, \phi, Y, s')$ s.t $\mathcal{Z}_1 \wedge \phi \neq$ $\emptyset$){
13        //$\mathcal{Z}_2$ is the successor
14        $\mathcal{Z}_2$ = $Extra^+_{LU}$(post($\mathcal{Z}_1, e$));
15        $E_{ZG}$ = $E_{ZG}$ ∪ $\{e\}$;
16        if($\exists(s', \mathcal{Z}_3) \in$ Q  $s.t$  $\mathcal{Z}_2 \subseteq \mathcal{Z}_3$){
17          $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{e}_{ZG} (s', \mathcal{Z}_3)\}$;
18        }
19        else{
20          $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{e}_{ZG} (s', \mathcal{Z}_2)\}$;
21          Q = Q $\cup \{(s', \mathcal{Z}_2)\}$;
22          D = D $\cup \{(s', \mathcal{Z}_2)\}$;
23         }
24      }
25      $\mathcal{Z}_2$ = $Extra^+_{LU}((\mathcal{Z}_1 \uparrow \wedge Inv(s)))$;
26      if($\exists(s, \mathcal{Z}_3) \in$ Q  $s.t$  $\mathcal{Z}_2 \subseteq \mathcal{Z}_3$){
27       $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{\epsilon}_{ZG} (s, \mathcal{Z}_3)\}$;
28      }
29      else{
30       $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{\epsilon}_{ZG} (s', \mathcal{Z}_2)\}$;
31       Q = Q $\cup \{(s, \mathcal{Z}_2)\}$;
32       D = D $\cup \{(s, \mathcal{Z}_2)\}$;
33      }
34   }
35  return MZG$(Q, q_0, (\Sigma \cup \{\epsilon\}), T_{ZG})$;
36  }
```

Algorithm 7.1: Reachable Multi-timed Zone Graph with Subsumption.

stored in Q and D (lines 30-32). Finally, in line 36, the algorithm returns a reachable zone graph.

Example 26. *Consider the two MTA $\mathscr{A}$ and $\mathscr{B}$ in Figure 7.8 with the finite input alphabet $\Sigma = \{a, b, c\}$, the set of processes $Proc = \{p, q\}$, the set of clocks $X_{\mathscr{A}} = \{x^p, y^q\}$, $X_{\mathscr{B}} = \{w^p, z^q\}$.*

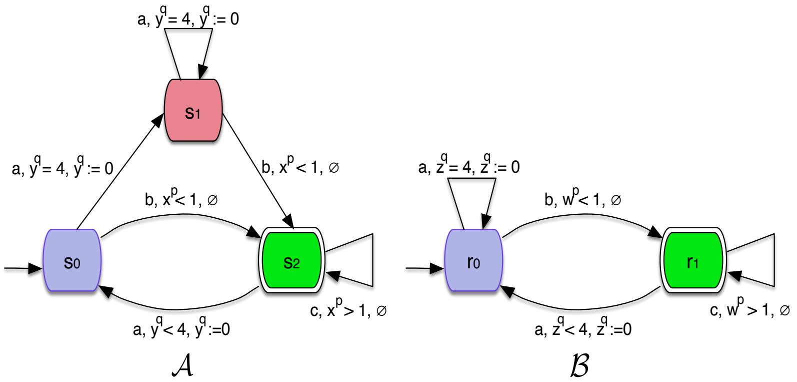
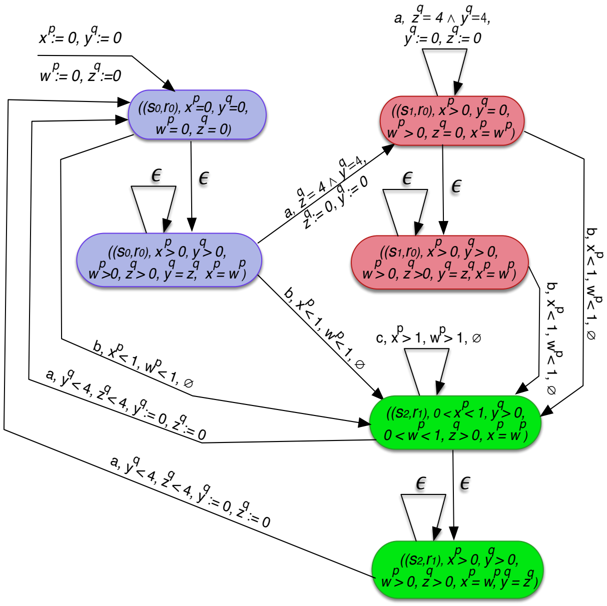


Figure 7.8: Two MTA $\mathscr{A}$ and $\mathscr{B}$

proposition 16 (Completeness)**.** *Let $\theta = (s_0, v_0, t_0) \xrightarrow{\vec{d}_0, a_0} (s_1, v_1, t_1) \xrightarrow{\vec{d}_1, a_1} (s_2, v_2, t_2) \dots \xrightarrow{\vec{d}_{n-1}, a_{n-1}} (s_n, v_n, t_n)$ be an initial (but not necessarily accepting) run of MLTS$(\mathscr{A}, \tau)$, for some $\tau \in Rates$. Then, for any state (s_i, v_i, t_i), where $0 \le i \le n$, appearing in this run, there exists a symbolic zone $(s_i, \mathcal{Z}_i)$ added in Q such that $v_i \in \mathcal{Z}_i$.*

Proof. We proceed by induction on the length of the run leading to (s_i, v_i, t_i).
Base case: We know that $v_0 \in \mathcal{Z}_0$. The zone $(s_0, Extra^{+}_{LU_{(s_0)}}(\mathcal{Z}_0))$ is added to D and Q in line 9.
Induction case: Assume that for all $0 \le i \le m$, there exists $(s_i, \mathcal{Z}_i)$ in Q such that $v_i \in \mathcal{Z}_i$. We will now show that there exists $(s_{m+1}, \mathcal{Z}_{m+1})$ in Q such that $v_{m+1} \in \mathcal{Z}_{m+1}$. By the induction hypothesis, we have $(s_m, \mathcal{Z}_m)$ in Q such that $v_m \in \mathcal{Z}_m$. Consider the transition $(s_m, v_m, t_m) \xrightarrow{\vec{d}_m, a_m} (s_{m+1}, v_{m+1}, t_{m+1})$ of the run θ and let e_m be its transition. As $(s_m, \mathcal{Z}_m)$ is in Q, the delay transition $\xrightarrow{\epsilon}_{\mathsf{ZG}}$ has been considered in the line 25 and represents $d_m > 0$. The other case $\vec{d}_m = 0$ means that $v' = v$ and is thus already included in $\mathcal{Z}_m$. Let $v^{'}_m = v_m + \vec{d}_m$. Then, since $(s_m, v_m, t_m) \xrightarrow{d_m} (s_m, v^{'}_m, t^{'}_m)$ is a delay transition of the MLTS$(\mathscr{A}, \tau)$, we have the time successor (i.e., $\mathcal{Z}_m = \mathcal{Z}_2 \leftarrow Extra^{+}_{LU}((\mathcal{Z}_1 \upharpoonleft) \wedge Inv(s)))$ and thus at line 27 we have added $\mathcal{Z}^{'}_m = \mathcal{Z}_3$ or $\mathcal{Z}^{'}_m = \mathcal{Z}_2$ at line 30, but in any case $v^{'}_m \in \mathcal{Z}_2 \subseteq Extra^{+}_{LU}(\mathcal{Z}_2) \subseteq \mathcal{Z}_3$. Let $(s_m, \mathcal{Z}^{'}_m) \xrightarrow{e_m}_{\mathsf{ZG}} (s_{m+1}, \mathcal{Z}_{m+1})$ be the discrete transition in the zone graph in lines 14 and 17. Let $(s_m, \mathcal{Z}^{'}_m) \xrightarrow{e_m}_{\mathsf{ZG}} (s_{m+1}, \mathcal{Z}_{m+1})$ be the transition in the zone graph in lines 26 and 27. By definition of the symbolic transition, $v_{m+1} \in \mathcal{Z}_{m+1}$. If $(s_{m+1}, \mathcal{Z}_{m+1})$ is in Q, we are done. The only other case when $(s_{m+1}, \mathcal{Z}_{m+1})$ is not in Q is when there exists $(s_{m+1}, \mathcal{Z}^{'}_{m+1})$ in Q such that $\mathcal{Z}_{m+1} \subseteq \mathcal{Z}^{'}_{m+1}$. Therefore, $v_{m+1} \in \mathcal{Z}_{m+1}$ and since $(s_{m+1}, \mathcal{Z}^{'}_{m+1})$ is in Q, our required zone would be $(s_{m+1}, \mathcal{Z}^{'}_{m+1})$. □

The proposition above tells us that Algorithm 7.1 over-approximates reachability. Now we can determine the termination of Algorithm 7.1 because there are finitely many $Extra^{+}_{LU}$ zones [BBLP06]. Here we will use Algorithm 7.1 to over-approximate the co-reachable state space of the two MTA $\mathscr{A}$ and $\mathscr{B}$, on the strongly synchronized product of $\mathscr{A}$ and $\mathscr{B}$ (i.e, $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$), as a precursor to our multi-timed bisimulation algorithm (see Section 7.4.4).



$x^p := 0,\ y^q := 0$
$w^p := 0,\ z^q := 0$
$((s_0,r_0),\ x^p=0,\ y^q=0,\ w^p=0,\ z^q=0)$
ϵ
ϵ
$((s_0,r_0),\ x^p>0,\ y^q>0,\ w^p>0,\ z^q>0,\ y^q=z^q,\ x^p=w^p)$
$a,\ z^q=4 \wedge y^q=4,\ y^q:=0,\ z^q:=0$
$((s_1,r_0),\ x^p>0,\ y^q=0,\ w^p>0,\ z^q=0,\ x^p=w^p)$
$a,\ z^q=4 \wedge y^q=4,\ z^q:=0,\ y^q:=0$
ϵ
ϵ
$((s_1,r_0),\ x^p>0,\ y^q>0,\ w^p>0,\ z^q>0,\ y^q=z^q,\ x^p=w^p)$
$b,\ x^p<1,\ w^p<1,\ \varnothing$
$b,\ x^p<1,\ w^p<1,\ \varnothing$
$b,\ x^p<1,\ w^p<1,\ \varnothing$
$c,\ x^p>1,\ w^p>1,\ \varnothing$
$b,\ x^p<1,\ w^p<1,\ \varnothing$
$a,\ y^q<4,\ z^q<4,\ y^q:=0,\ z^q:=0$
$((s_2,r_1),\ 0<x^p<1,\ y^q>0,\ 0<w^p<1,\ z^q>0,\ x^p=w^p)$
ϵ
ϵ
$a,\ y^q<4,\ z^q<4,\ y^q:=0,\ z^q:=0$
$((s_2,r_1),\ x^p>0,\ y^q>0,\ w^p>0,\ z^q>0,\ x^p=w^p,\ y^q=z^q)$

Figure 7.9: The multi-timed zone graph based on the composition of $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$.

Example 27. *Consider the zone graph in Figure 7.9 which was build from the parallel composition of the two MTA $\mathscr{A}$ and $\mathscr{B}$ in Figure 7.8.*

7.4.4 Refinement Algorithm

In algorithm 7.2 we describe a refinement algorithm (see Section 5.3.10) to compute the multi-timed bisimulation from its zone graph $\mathsf{MZG}(\mathscr{C})$, where $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$. The state space Q of $\mathsf{MZG}(\mathscr{C})$ is initially divided into zones that over-approximate the co-reachable states of $\mathscr{A}$ and $\mathscr{B}$. In essence, our algorithm 7.2 is based on the refinement technique [PT87] [TY01] [BO02], but with some variations due to independent clocks where delay transitions (unlike in the case of Figure 5.12, see section 5.3.10) can continuously traverse diagonal, nearly vertical, and horizontal time successor zones. Conversely, when the clocks are perfectly synchronous, the delay transition traverses only continuously diagonal time successor zones. Thus, the time refinement operator presented in [TY01] has been adapted to our algorithm 7.2. Figure 7.10 shows an example of delay transitions traversing continuously diagonal, nearly horizontal, and vertical time successor regions for two independent local clocks $X = \{x^p, y^p\}$. We define a multi-time refinement operator that perfectly matches the characteristics of our delay transitions. The discrete refinement operator presented in [TY01] is also not applicable within our algorithm 7.2, because it can be applied over models with synchronized clocks (i.e., clocks evolving at the same rate) and single time steps. Therefore, our algorithm 7.2 assigns states to equivalence zones according to their common actions. In each refinement iteration, the set of zones is refined according to their actions. The algorithm in [PT87], cannot be applied directly in our setting due to its untimed nature, while in our case delay and discrete transitions should be considered. Based on [TY01], we introduce a discrete and multi-time refinement operator that refines the set of zones until a fixed point is reached, which is the multi-timed bisimilarity. Thus, we introduce the multi-time and discrete refinement operators.

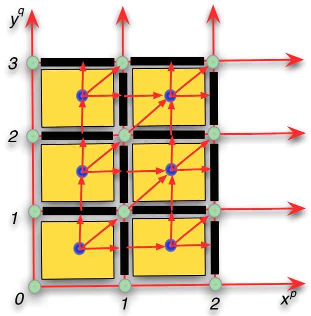


Figure 7.10: Multiple time successors

Definition 87 (**Multi-Time Refinement Operator**)**.** *Let $q = (s, \mathcal{Z})$ and $q' = (s, \mathcal{Z}')$ be two zones of the same location, then :*

$$TimePred_{\uparrow}(\mathcal{Z}, \mathcal{Z}') = \{v \in \mathcal{Z} \mid \exists\, \vec{d} \in \mathbb{R}^{Proc}_{>0},\ \exists\, \tau \in Rates,\ \exists\, t, t'' \geq 0,\ \ t \leq t'',$$

$$\vec{d} = \tau(t'') - \tau(t),\ (v +_{\pi} \vec{d}) \in\ \mathcal{Z}',\ and\ \forall t', t \leq t' \leq t'',$$

$$\exists \vec{d'}, \vec{d'} = \tau(t') - \tau(t)\ then\ (v +_{\pi} \vec{d'}) \in\ (\mathcal{Z} \cup \mathcal{Z}')\}$$

Then, $TimePred_{\uparrow}(\mathcal{Z}, \mathcal{Z}')$ is the set of valuations in the zone $\mathcal{Z}$ from which a valuation of $\mathcal{Z}'$ can be reached through the elapsing of time, while staying in $(\mathcal{Z} \cup \mathcal{Z}')$ without entering any other zones besides $\mathcal{Z}$ and $\mathcal{Z}'$ (i.e., $\mathcal{Z} \cup \mathcal{Z}'$). The $TimePred_{\uparrow}(\mathcal{Z}, \mathcal{Z}')$ operator refines $\mathcal{Z}$ by selecting the states that can reach $\mathcal{Z}'$.

proposition 17. *Let $q = (s, \mathcal{Z})$, $q' = (s, \mathcal{Z}') \in Q$ be two zones, then $TimePred_{\uparrow}$ $(\mathcal{Z}, \mathcal{Z}')$ is a clock zone.*

Our proof follows the same lines as the proof of [TY01] (p.48).

Proof. Let $q = (s, \mathcal{Z})$, $q' = (s, \mathcal{Z}')$ and $\mathcal{Z}'' = \mathrm{TimePred}_{\uparrow}(\mathcal{Z}, \mathcal{Z}')$. We show that $\mathcal{Z}''$ is convex, i.e, if v_1, $v_2 \in \mathcal{Z}''$ then $v = kv_1 + (1-k)v_2 \in \mathcal{Z}''$, for $0 < k < 1$. v_1, $v_2 \in \mathcal{Z}''$ implies that v_1, $v_2 \in \mathcal{Z}$ and $\exists\ \vec{d_1}, \vec{d_2} \in \mathbb{R}^{Proc}_{>0}$ such that $v_1 +_{\pi} \vec{d_1}$, $v_2 +_{\pi} \vec{d_2} \in \mathcal{Z}'$ and $\exists\ t_1$, $t_2 \geq 0$, $t_1 \leq t_2$, $\forall\ t'$, $t_1 \leq t' \leq t_2$, $\vec{d_1} = \tau(t_2) - \tau(t')$ and $\vec{d_2} = \tau(t') - \tau(t_1)$ then $v_1 +_{\pi} \vec{d_1} \in \mathcal{Z}'$, $v_2 +_{\pi} \vec{d_2} \in (\mathcal{Z} \cup \mathcal{Z}')$. Let $\vec{d} = k\vec{d_1} + (1-k)\vec{d_2}$, then $v +_{\pi} \vec{d} =$

$k(v_1 +_\pi \vec{d}_1) + (1-k)(v_2 +_\pi \vec{d}_2)$, implying that $v +_\pi \vec{d} \in \mathcal{Z}'$, since $\mathcal{Z}'$ is zone. Now, we have to show that $\forall\ t'$, $t_1 \le t' \le t_2$, $\vec{d} = \tau(t_2) - \tau(t')$ and $\vec{d}' = \tau(t') - \tau(t_1)$, $v +_\pi \vec{d}'$ $\in (\mathcal{Z} \cup \mathcal{Z}')$. Given $\vec{d}'$, $\vec{d}$, we can write $\vec{d}'$ as $k\vec{d}_3 + (1-k)\vec{d}_4$, for some $\vec{d}_3$, $\vec{d}_1$ and $\vec{d}_4$, $\vec{d}_2$. We have $v_1 +_\pi \vec{d}_3$, $v_2 +_\pi \vec{d}_4 \in (\mathcal{Z} \cup \mathcal{Z}')$. If both $v_1 +_\pi \vec{d}_3$, $v_2 +_\pi \vec{d}_4 \in \mathcal{Z}$ or $v_1 +_\pi \vec{d}_3$, $v_2 +_\pi \vec{d}_4 \in \mathcal{Z}'$, we are done, since $\mathcal{Z}$ and $\mathcal{Z}'$ are both zones. Considerer the case $v_1 +_\pi \vec{d}_3 \in \mathcal{Z}$ and $v_2 +_\pi \vec{d}_4 \in \mathcal{Z}'$. Let g be the smallest positive real such that $v_2 +_\pi \vec{d}_4 - g \in \mathcal{Z}$ or $v_1 +_\pi \vec{d}_3 - g(1-\frac{1}{k}) \in \mathcal{Z}'$. To assume the first case, we have $v_1 +_\pi \vec{d}_5$, $v_2 +_\pi \vec{d}_6 \in \mathcal{Z}$, for $\vec{d}_6 = \vec{d}_4 - g$ and $\vec{d}_5 = \vec{d}_3 - g(1-\frac{1}{k})$. Moreover, $\vec{d}' = k\vec{d}_5 + (1-k)\vec{d}_6$, which means that $v +_\pi \vec{d}' = k(v_1 +_\pi \vec{d}_5) + (1-k)(v_2 +_\pi \vec{d}_6)$. By convexity of $\mathcal{Z}$, $v +_\pi \vec{d}'$ $\in \mathcal{Z}$. □

For the discrete refinement operator, we use as the stamp of a state (s, v) (i.e., states in MLTS $(\mathscr{A}, \tau)$, where $\mathscr{A}$ is a MTA) the set of outgoing discrete transitions from (s', v') with the same actions. Then, a refinement of a zone can be computed by grouping states that have the same actions. The resulting set of zones then represents the multi-timed bisimulation relation: two states (s, v) and (s', v') are multi-timed bisimilar iff they are in the same zone with similar outgoing transitions (i.e., the same actions). Formally, this is captured in the following definition:

Definition 88 (Action Refinement Operator). *Let $q = (s, \mathcal{Z})$ be a zone, then the stamp of a state $(s, v) \in q$ formed by the set of labels of all the edges starting from $(s, v) \in S \times \mathbb{R}^X_{\ge 0}$ is defined as: $ActionSigPred_q(s, v) = \{a \mid \exists\ v' \in \mathcal{Z}, \exists\ s' \in S,\ (s, v) \xrightarrow{a}_{mlts} (s', v')\}$. Also, the stamp of the zone q is defined as:*

$$ActionSig(q) = \bigcup_{(s,v)\in q} ActionSigPred_q(s, v)$$

The $\mathsf{ActionSigPred}_q(s, v)$ operator is used to compute the stamps of a state in a zone. Our algorithm 2 consists of two steps: The initial phase, is responsible for keeping pairs of states into zones so that every pair of states $((s_\mathscr{A}, s_\mathscr{B}),$ $(v_\mathscr{A}, v_\mathscr{B}))$ from the same zone q have the same action $\mathsf{ActionSigPred}_q\ (s_\mathscr{A}, v_\mathscr{A}) =$ $\mathsf{ActionSigPred}_q(s_\mathscr{B}, v_\mathscr{B})$. The refinement phase consists of computing the timed predecessors (see Definition 89 below) and the discrete action predecessors (see Definition 90 below) until a stable set of zones is reached. Stable zones are a multi-timed bisimulation relation only if each pair of states of each zone in the set has the same action with respect to each computed refinement. A detailed explanation of our algorithm 2 follows:

7.4.4.1 Initial phase

Let $\Pi_0 = Q$ be the initial set of zones, where Q is given by algorithm 1. After the instruction $(\Pi \leftarrow \Pi_0)$ in line 6, the set Π contains zones consisting of states with unique actions: Π is composed of zones $((s_\mathscr{A}, s_\mathscr{B}), \mathcal{Z})$, where $\mathsf{ActionSigPred}_q(s_\mathscr{A}, v_\mathscr{A}) =$ $\mathsf{ActionSigPred}_q(s_\mathscr{B}, v_\mathscr{B})$ for any $(v_\mathscr{A} \cup v_\mathscr{B}) \in \mathcal{Z}$. Let us recall that for simplicity, we will write $(s, \mathcal{Z})$ to denote the pairs $((s_\mathscr{A}, s_\mathscr{B}), \mathcal{Z})$.

7.4.4.2 Refinement phase

The set of zones is iteratively refined until all zones becomes stable with respect to all their timed predecessors and discrete predecessors.

Definition 89 (Time Refinement). *Let* Π *be a set of zones and* $q = (s, \mathcal{Z})$*,* $q' = (s', \mathcal{Z}')$ *be two zones in* Π *with* $s = s'$*. Then for the delay transitions, the refinement function is defined as follows:*

$TimeRefines(\mathcal{Z}, \Pi) = \{TimePred_{\uparrow}(\mathcal{Z}, \mathcal{Z}') \mid \mathcal{Z}' \in \Pi, q \xrightarrow{\epsilon}_{ZG} q'\}$.

Definition 90 (Discrete Refinement). *Let* Π *be a set of zones and* $q = (s, \mathcal{Z})$*,* q' $= (s', \mathcal{Z}')$ *be two zones in* Π*. Let* $q = (s, \mathcal{Z})$ *be the currently examined zone and* $ActionSig(q)$ *be the actions of the set of states into the zone* q*. Let* $e_{\mathscr{A}}$ *and* $e_{\mathscr{B}}$ *be transitions of the MTA* $\mathscr{A}$ *and* $\mathscr{B}$*. Then the refinement of a zone* q *is defined as follows:*

$DiscreteSigRefine(\mathcal{Z}, \Pi) = \bigcap_{a \in ActionSig(q)} ((\bigcap_{\{e_{\mathscr{A}} | Action(e_{\mathscr{A}})=a\}} \bigcup_{\{e_{\mathscr{B}} | Action(e_{\mathscr{B}})=a\}} pred(\mathcal{Z}', (e_{\mathscr{A}}, e_{\mathscr{B}}))) \cap (\bigcap_{\{e_{\mathscr{B}} \mid Action(e_{\mathscr{B}})=a\}} \bigcup_{\{e_{\mathscr{A}} \mid Action(e_{\mathscr{A}})=a\}} pred(\mathcal{Z}', (e_{\mathscr{A}}, e_{\mathscr{B}}))))$.

proposition 18. *Let* $(s, \mathcal{Z})$ *be a zone of* Π *and let* $(e_{\mathscr{A}}, e_{\mathscr{B}})$ *be an edge of the* $MZG(\mathscr{C})$*, then each of* $TimeRefine(\mathcal{Z}, \Pi)$ *and* $DiscreteSigRefine(\mathcal{Z}, \Pi)$ *forms a stable set of* $\mathcal{Z}$ *in* Π*.*

Our proof follows the same lines as the proof of [TY01].

Proof.

(i) Consider $\Pi_1 = \mathsf{TimeRefine}(\mathcal{Z}, \Pi)$. By Lemma 17, all members of Π are zones. It remains to show that their union yields $\mathcal{Z}$. Let $\mathcal{Z}_i \in \Pi_1$, $\mathcal{Z}_i = \mathsf{TimePred}_{\uparrow}(\mathcal{Z}, \mathcal{Z}'_i)$, where $\mathcal{Z}'_i \in \Pi$, for $i = 1, 2$. Since Π is a stable set of zones, $\mathcal{Z}$, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are all disjoint. Assumes $v \in \mathcal{Z}_1 \cap \mathcal{Z}_2$. For $i = 1, 2$, $\exists\, \vec{d}_i \in \mathbb{R}^{Proc}_{>0}$ such that $v + \vec{d}_i \in \mathcal{Z}'_i$ and $\forall v \in \mathcal{Z}$, $\exists \vec{d}_i \in \mathbb{R}^{Proc}_{>0}$, $\exists\, \tau \in Rates$, $\exists\, t, t'' \geq 0$ and $t \leq t''$, $\forall t', t \leq t' \leq t''$, $(v + \vec{d}_i) \in \mathcal{Z}'_i$, $\vec{d}_i = \tau(t'') - \tau(t)$ and $\forall\, \vec{d}'_i$, $0 \leq \vec{d}'_i \leq \vec{d}_i$ then $(v + \vec{d}'_i) \in (\mathcal{Z} \cup \mathcal{Z}'_i)$, $\vec{d}'_i = \tau(t') - \tau(t)$. Observe that $\vec{d}_1 \neq \vec{d}_2$, since $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are disjoint. Without loss of generality, assume $\vec{d}_1 < \vec{d}_2$. We have that $v + \vec{d}_1 \in \mathcal{Z}'_1$ and $v + \vec{d}_1 \in \mathcal{Z} \cup \mathcal{Z}'_2$, that is, either $v + \vec{d}_1 \in \mathcal{Z}'_1 \cap \mathcal{Z}$ or $v + \vec{d}_1 \in \mathcal{Z}'_1 \cap \mathcal{Z}'_2$, which contradicts the fact that $\mathcal{Z}$, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are all disjoint. This proves that $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are disjoint. Now, let $v \in \mathcal{Z}$. We can find $\mathbb{R}^{Proc}_{>0}$ and $\mathcal{Z}' \in \Pi$ such that $v + \vec{d}$ $\in \mathcal{Z}'$ and $\forall v \in \mathcal{Z}$, $\exists \vec{d} \in \mathbb{R}^{Proc}_{>0}$, $\exists\, \tau \in Rates$, $\exists\, t, t'' \geq 0$ and $t \leq t''$, $\forall t', t \leq t' \leq$ t'', $(v + \vec{d}) \in \mathcal{Z}'$, $\vec{d} = \tau(t'') - \tau(t)$ and $\forall\, \vec{d}'$, $0 \leq \vec{d}' \leq \vec{d}$ then $(v + \vec{d}') \in (\mathcal{Z} \cup \mathcal{Z}')$, $\vec{d}' = \tau(t') - \tau(t)$. By definition, $v \in \mathsf{TimePred}_{\uparrow}(\mathcal{Z}, \mathcal{Z}')$.

(ii) Now, consider $\Pi_2 = \mathsf{DiscreteSigRefine}(\mathcal{Z}, \Pi)$. For all members of Π_2 are zones. By the distributivity of pred over union $(\mathsf{pred}(\mathcal{Z}_1 \cup \mathcal{Z}_2, e) = \mathsf{pred}(\mathcal{Z}_1, e) \cup \mathsf{pred}(\mathcal{Z}_2, e))$ is a zone of $\mathcal{Z}$. It remains to show that they are disjoint. Let $\mathcal{Z}_i \in \Pi_2$, $\mathcal{Z}_i = \mathcal{Z} \cap \mathsf{pred}(\mathcal{Z}'_i, e)$, where $\mathcal{Z}'_i \in \Pi$, for $i = 1, 2$. Since Π is a a stable set of zones, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are disjoint. Assume $v \in \mathcal{Z}_1 \cap \mathcal{Z}_2$. Recall that the successor of v, say v', is unique. Since $v \in \mathsf{pred}(\mathcal{Z}'_1, e) \cap \mathsf{pred}(\mathcal{Z}'_2, e)$, it must be that $v' \in \mathcal{Z}'_1 \cap \mathcal{Z}'_2$, which contradicts $\mathcal{Z}'_1 \cap \mathcal{Z}'_2 = \emptyset$.

□

```
Input: A MZG($\mathscr{C}$)= $(Q, q_0 = (q^0_{\mathscr{A}}, q^0_{\mathscr{B}}), \Sigma = \Sigma_{\mathscr{A}} = \Sigma_{\mathscr{B}}, \rightarrow_{ZG})$, $\Pi_0 = Q$
Output: A stable set of zones $\Pi$
//$\Pi$ is a set of zones, $\mathscr{Z}$ are clock zones

ZG PartitionZoneGraph(ZG $\mathscr{C}$, $\Pi_0$){
  //Get the input set of zones $\Pi$
  $\Pi'$ = $\Pi_0$;
  do{
   // Refine $\Pi'$ by delay transitions
   for(each zone  $\mathscr{Z} \in \Pi'$){
     $\Pi'$ = TimeRefine($\mathscr{Z}$, $\Pi'$);
    }
   // Refine $\Pi'$ by discrete transitions
   for(each zone  $\mathscr{Z} \in \Pi'$) {
    $\Pi' \leftarrow$ DiscreteSigRefine($\mathscr{Z}$, $\Pi'$);
   }
 }while($\Pi'$ does not change)
return $\Pi'$;
}
```

Algorithm 7.2: The Refinement Algorithm for a Reachable MZG.

In short, algorithm 7.2 starts with an initial set of zones Π_0 and successively refines this set such that each zone eventually contains only bisimilar state pairs. The definition TimeRefine($\mathscr{Z}$,Π) above generates a finer set of zones that deals with delay transitions. The definition of DiscreteSigRefine($\mathscr{Z}$,Π) above also generates a finer set of zones and distinguishes the states by their discrete transitions. Termination is ensured by the fact that at worst the refinement can yield the region automaton.

algorithm 7.2 describes the main steps of the decision procedure for multi-timed bisimulation checking. It uses the function BuildSymbZoneGraph (i.e., algorithm 7.1). The function PartitionZoneGraph returns a stable set of zones Π. Given a set of zones Π, algorithm 7.2 computes the states $((s_{\mathscr{A}}, s_{\mathscr{B}}), \mathscr{Z})$ from Π that are bisimilars in particular whether the initial state $((s^0_{\mathscr{A}}, s^0_{\mathscr{B}}), \mathscr{Z}_0)$ are bisimilar.

proposition 19. *Let Π be an initial set of zones and $q = (s, \mathscr{Z})$ be a zone in Π. Let Π' be a final stable set of zones. Let $(s_{\mathscr{A}}, \nu_{\mathscr{A}})$ and $(s_{\mathscr{B}}, \nu_{\mathscr{B}})$ be two states in q, then $(s_{\mathscr{A}}, \nu_{\mathscr{A}}) \approx (s_{\mathscr{B}}, \nu_{\mathscr{B}})$ iff $((s_{\mathscr{A}}, s_{\mathscr{B}}), \nu_{\mathscr{A}} \cup \nu_{\mathscr{B}}) \in q'$, where q' is a zone of the final set of stable zones Π'.*

Now we show that the problem of deciding whether two MTA are multi-timed bisimilar is EXPTIME-complete. Our algorithm 7.2 uses a reachable multi-timed zone graph ZG($\mathscr{C}$) to decide multi-timed bisimilarity. Our approach is based on a simple reduction of Linearly Bounded Alternating Turing Machines (LB-ATM) [AL99]. Our reduction can be applied to both TA and MTA. Timed bisimulation has been shown to be decidable for TA in EXPTIME. [AL99] [HHK95] [WL97] [LS00]. We reduce the acceptance problem for LB-ATM to the problem of deciding whether two MTA are multi-timed bisimilar.

Theorem 59. *Deciding multi-timed bisimulation between two MTA is EXPTIME-complete.*

Proof. EXPTIME-hardness: The proof follows from the EXPTIME-hardness of timed bisimilarity on TA [AL99] [LS00], as MTA are an extension of TA: If we use a single process, MTA = TA. □

Proof. EXPTIME-membership: EXPTIME-membership can be deduced from the EXPTIME-membership of the same problem for TA [AL99] [WL97]. The EXPTIME-membership can be obtained by applying decision algorithms over the reachable multi-timed zone graph corresponding to the parallel composition of two MTA. Let $\mathscr{C}$ be a parallel composition of two MTA $\mathscr{A}$ and $\mathscr{B}$. Let q be a state of $\mathscr{C}$, deciding whether two states $(s_{\mathscr{A}}, v_{\mathscr{A}})$ and $(s_{\mathscr{B}}, v_{\mathscr{B}})$ in q are multi-timed bisimilar can be done by using the decision algorithm in [WL97]. Thus the number of zones in the multi-timed zone graph is exponential in the number of clocks of the MTA $\mathscr{A}$. Hence, decide multi-timed bisimilarity is done in EXPTIME. □

Example 28. *An example of a stable set of zones computed by our algorithms can be found in Figure 7.11. Figure 7.11 shows the multi-timed bisimilar graph from Figure 7.9.*

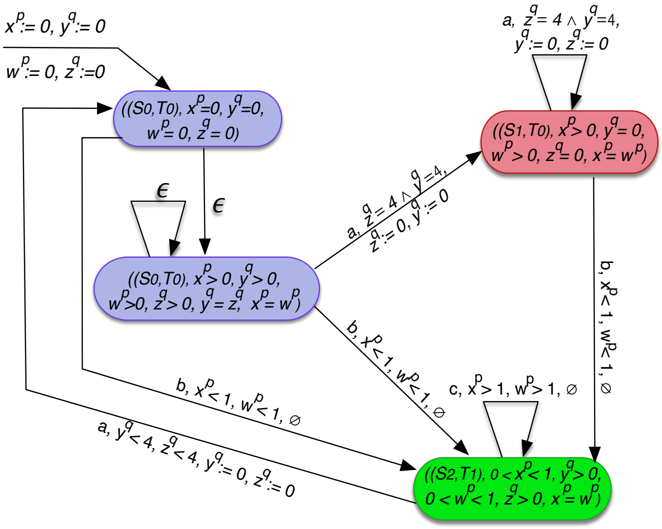

x^p:= 0, y^q:= 0
w^p:= 0, z^q:=0
((S0,T0), x^p=0, y^q=0, w^p= 0, z^q= 0)
ϵ
ϵ
((S0,T0), x^p> 0, y^q> 0, w^p>0, z^q> 0, y^q = z^q, x^p= w^p)
a, z^q= 4 ∧ y^q=4, y^q:= 0, z^q:= 0
((S1,T0), x^p> 0, y^q= 0, w^p> 0, z^q= 0, x^p= w^p)
a, z^q= 4 ∧ y^q=4, z^q:= 0, y^q:= 0
b, x^p< 1, w^p< 1, ∅
b, x^p< 1, w^p< 1, ∅
c, x^p> 1, w^p> 1, ∅
b, x^p< 1, w^p< 1, ∅
a, y^q< 4, z^q< 4, y^q:= 0, z^q:= 0
((S2,T1), 0 < x^p< 1, y^q> 0, 0 < w^p< 1, z^q> 0, x^p= w^p)

Figure 7.11: Multi-timed bisimulation result from $\mathscr{A}$ and $\mathscr{B}$ in Figure 7.8

7.5 A Multi-Timed Modal Logic

Extensions of timed modals and temporal logics, such as Timed Propositional Modal Logic (TPML), μ-calculus [HNSY94], L_ν [LLW95], and Timed Computation Tree Logic (TCTL [TY01]) have been used to specify sequential (mono-timed) systems whose behavior is governed by timing constraints. However, in these logics, the information about independent clocks and distributed components observed in a DRTS is modeled in a global setting [Ray15]. Consequently, these logics may not be suitable for explicitly specifying local timing properties that need to hold only in selected parts of the overall system. In essence, this means that the mono-timed semantics of these logics is defined in terms of Timed Labelled Transition Systems (TLTS). However, there are logics that have been defined to capture aspects of distributed components and timing properties of DRTS: e.g., DRTL [MP90], APTL [WME93] and DECTL [OLS11], among others. Roughly speaking, these logics allow the definition of formulas whose truth values depend on (or are relative to) only part of their underlying mathematical models. In the case of DRTL and APTL, these logics are an extension of Second Order Logic (SOL) and First Order Logic (FOL), where the set of formulas consists of constants, functions, predicates, universal and existential quantifiers, and logical connectives. In general, these logics are undecidable, but depending on which fragment is used, the resulting logic may be decidable. In the case of DECTL, a distributed real-time logic with independent time evolutions is proposed. This logic can be model-checked by translating a DECTL formula into a distributed event clock automaton [OLS11]. In general, this timed temporal logic does not use different action labels and delays, i.e. it is interpreted over TLTS. Unlike timed temporal logics, timed modal logics distinguish transitions of a TLTS with different actions and delays. On the other hand, timed modal logics can be used to study behavioral equivalence by using bisimulation [Mil89]. Timed bisimulation [Cer93] was used to verify the preservation of mono-timed behavior and the timed properties expressed in TCTL and L_ν. Timed bisimulation was shown to be decidable for Timed Automata (TA) [AD94] [LLW95]. Based on MLTS, which is an extension of TLTS to deal with the notion of distributed clocks, we propose ML_ν, an extension of L_ν that relies on a distributed semantics for Timed Automata (TA). Instead of considering uniform clocks over the distributed systems, we let time vary independently in each TA. We define the syntax and semantics of ML_ν over executions of MLTS with such a semantics, and we show that its model checking problem against ML_ν is EXPTIME-complete.

7.5.1 Syntax of ML_ν

We define ML_ν, a multi-timed modal logic that extends the logic L_ν over distributed clocks. We first present the syntax of the logic ML_ν.

Definition 91. *Let Σ be a finite alphabet, X be a finite set of clocks, $Proc$ be a set of processes, $\pi : X \to Proc$ be a function that maps each clock to a process and Id the set of proposition identifiers. The formulae of ML_ν over Σ, X and Id are defined by the grammar:*

$$\varphi ::= \; true \mid false \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a]\varphi \mid \langle a \rangle \varphi \mid$$

$$\exists\varphi \mid \forall\varphi \mid x^p \; \underline{in} \; \varphi \mid \phi \mid \; x^p + c \sim y^p + d \mid Z$$

where $a \in \Sigma$, $p \in Proc$, $x^p, y^p \in X$, c, $d \in \{0, \cdots, k\}$, k a non-negative integer, $\sim \in \{=, >, \geq, <, \leq\}$, $Z \in Id$, $\phi \in \Phi(X)$ a clock constraint, $[a]\varphi$, $\langle a \rangle \varphi$ are two modalities of the logic, and $\exists\varphi$ and $\forall\varphi$ are the two timed modalities. Note that we can only compare two clocks of the same process.

The identifiers Id are specified by a declaration environment $\mathscr{D}$ assigning a ML_ν formula to every identifier in order to define properties with maximal fixpoints. A declaration is noted by $Z \overset{\text{def}}{=} \varphi$ for $\mathscr{D}(Z) = \varphi$.

7.5.2 Semantics of ML_ν

Let $\mathscr{A}$ be a MTA over *Proc* and $\tau \in$ Rates and assume that $\mathsf{MLTS}(\mathscr{A}, \tau) = (Q, q_0, \Sigma, \rightarrow_{mlts})$ gives its semantics. Now, we interpret L_ν formulas over extended states. An extended state over Q is a pair (q, μ), where $q \in Q$ is a MLTS state (Definition 5) and μ a valuation for the formula clocks in X. An extended state satisfies an identifier Z if it belongs to the maximal fixpoint of the equation $Z = \mathscr{D}(Z)$. The formal semantics of ML_ν formulas interpreted over $\mathsf{MLTS}(\mathscr{A}, \tau)$ is given by the satisfaction relation $\models$, defined as the largest relation that satisfies the equivalences in 91.

The intuition for the different operators is as follows. The formula $x^p \; \underline{in} \; \varphi$ introduces a formula clock x^p for $p \in Proc$ and initializes it to 0; i.e. an extended state satisfies the formula if and only if the modified state, with x^p reset to 0, satisfies φ. The semantics of the *diamond* modality $\langle a \rangle \varphi$ is, informally, that in a given state it is possible to perform an a action and go to a state where φ holds, and double for the *box* modality $[a]\varphi$, all transitions labeled by a go to such a state. Informally, $\exists\varphi$ (resp. $\forall\varphi$) holds in an extended state if there exists (resp. if every) delay transition leading to a state satisfying φ. $\exists$ (resp. $\forall$) denotes existential (resp. universal) quantification over (arbitrary) delay transitions. Dual $\forall\varphi$ holds in a state if every delay transition leads to a state satisfying φ. $\forall$ denotes universal quantification over delay transitions.

Definition 92. *Let Σ be a finite alphabet, X be a finite set of clocks and $Proc$ be a set of processes. The semantics of formulae in ML_ν is implicitly given with respect to a given MLTS inductively as follows:*

$$\begin{array}{lcl}
(q,\mu) \models & true & \Leftrightarrow true \\
(q,\mu) \models & false & \Leftrightarrow false \\
(q,\mu) \models & \varphi_1 \wedge \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \; and \; (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \varphi_1 \vee \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \; or \; (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \phi & \Leftrightarrow \mu \models \phi \; for \; \phi \in \Phi(X) \\
(q,\mu) \models & [a]\varphi & \Leftrightarrow \forall q \xrightarrow{a}_{mlts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & \langle a \rangle \varphi & \Leftrightarrow \exists q \xrightarrow{a}_{mlts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & x^p \; \underline{in} \; \varphi & \Leftrightarrow (q, \mu[x^p \to 0]) \models \varphi \\
(q,\mu) \models & \exists\varphi & \Leftrightarrow \exists \vec{d} \in \mathbb{R}_{\geq 0}^{Proc}, \exists q' \in Q, \; such \; that \; q \xrightarrow{\vec{d}}_{mlts} q', \\
 & & (q, \mu +_\pi \vec{d}) \models \varphi
\end{array}$$

$$\begin{array}{lll} (q,\mu) \models & \forall\varphi & \Leftrightarrow \forall \vec{d} \in \mathbb{R}_{\geq 0}^{Proc}, \forall q' \in Q, \textit{such that } q \xrightarrow{\vec{d}}_{mlts} q', \\ & & (q,\mu +_{\pi} \vec{d}) \models \varphi \\ (q,\mu) \models & x^p + c \sim y^p + d & \Leftrightarrow \mu(x^p) + c \sim \mu(y^p) + d \\ (q,\mu) \models & Z & \textit{the maximal fixpoint in } \mathcal{D}(Z) \end{array}$$

Two formulae are equivalent iff they are satisfied by the same set of extended states in every MLTS.

Definition 93. *A state q in a MLTS satisfies a formula φ, iff $(q,\mu_0) \models \varphi$ where μ_0 is the clock valuation that maps each formula clock to zero.*

Definition 94. *Let $\mathscr{A}$ be a MTA and $\varphi \in ML_v$, then $\mathscr{A} \models \varphi$ iff $\forall \tau \in Rates$, $MLTS(\mathscr{A},\tau) \models \varphi$.*

Let φ be a closed formula, then the set of extended states satisfying φ is independent of the valuation μ for the formula clocks. Thus, if φ is closed then for any state q in a MLTS and valuations μ, μ' for the formula clocks, we can get that $(q,\mu) \models \varphi$ iff $(q,\mu') \models \varphi$. Therefore, if φ is closed, it makes sense to talk about a state q that satisfies φ.

Theorem 60. *Let $Proc$ be a set of processes. Let $\mathscr{M} = (Q, q_0, \Sigma, \rightarrow_{mlts})$ be a MLTS and q_1, q_2 be multi-timed bisimilar states in Q. Let μ be a clock valuation for the formula clocks in X, then the extended states (q_1,μ) and (q_2,μ) satisfy exactly the same formulae in ML_v.*

Proof. Assume that q_1, q_2 are multi-timed bisimilar states in Q. Let μ be a clock valuation for the formula clocks in X. Assume that $(q_1,\mu) \models \varphi$ for some formula $\varphi \in ML_v$. Using structural induction on φ, we shall prove that $(q_2,\mu) \models \varphi$. By symmetry, this is enough to establish that (q_1,μ) and (q_2,μ) satisfy the same formulae in ML_v.

The proof proceeds by a case analysis on the form of φ. Here, we present the details only for four modalities $\forall\ \varphi_1$, the other modalities can be proved in the same way. Our inductive hypothesis is that, for all states r_1 and r_2, if r_1 and r_2 are multi-timed bisimilar and $(r_1,\mu') \models \varphi_1$ for some valuation μ' of the formula clocks, then $(r_2,\mu') \models \varphi_1$. Using this hypothesis, we shall prove that :

- $(q_2,\mu) \models \forall\ \varphi_1$: To this end, assume that, for every $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$, for all $q_2' \in Q$, $q_2 \xrightarrow{\vec{d}}_{mlts} q_2'$. We wish to show that $(q_2',\mu +_{\pi} \vec{d}) \models \varphi_1$. Now, since q_1 and q_2 are multi-timed bisimilar and $q_2 \xrightarrow{\vec{d}}_{mlts} q_2'$, there is a state $q_1' \in Q$, $q_1 \xrightarrow{\vec{d}}_{mlts} q_1'$ and q_1' is multi-timed bisimilar to q_2'. By our supposition that $(q_1,\mu) \models \varphi$, we have that $(q_1',\mu +_{\pi} \vec{d}) \models \varphi_1$. The inductive hypothesis yields that $(q_2',\mu +_{\pi} \vec{d}) \models \varphi_1$. Since q_2' and $\vec{d}$ were arbitrary we may conclude that $(q_2,\mu) \models \forall\ \varphi_1$, which was to be shown.
- $(q_2,\mu) \models \exists\ \varphi_1$: To this end, assume that, there is some $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$, there is some $q_2' \in Q$, $q_2 \xrightarrow{\vec{d}}_{mlts} q_2'$. We wish to show that $(q_2',\mu +_{\pi} \vec{d}) \models \varphi_1$. Now, since q_1 and q_2 are multi-timed bisimilar and $q_2 \xrightarrow{\vec{d}}_{mlts} q_2'$, there is a state $q_1' \in Q$, $q_1 \xrightarrow{\vec{d}}_{mlts} q_1'$ and q_1' is multi-timed bisimilar to q_2'. By our supposition that

$(q_1,\mu) \models \varphi$, we have that $(q'_1, \mu_\pi + \vec{d}) \models \varphi_1$. The inductive hypothesis yields that $(q'_2, \mu_\pi + \vec{d}) \models \varphi_1$. Since q'_2 and $\vec{d}$ were arbitrary we may conclude that $(q_2,\mu) \models \exists\, \varphi$, which was to be shown.

- $(q_2,\mu) \models y^p\ \underline{in}\ \varphi_1$. To this end, assume that, for some $y^p \in X$, for $p \in Proc$, such that $(q_2, \mu[y^p \to 0]) \models \varphi_1$. Now, since q_1 and q_2 are multi-timed bisimilar and by our supposition that $(q_1,\mu) \models y^p\ \underline{in}\ \varphi$ for some $y^p \in X$ for $p \in Proc$, we have that $(q_1, \mu[y^p \to 0]) \models \varphi$ reset y^p to 0.
- $(q_2,\mu) \models y^p + c \sim x^p + d$. To this end, assume that, for some y^p, $x^p \in X$ for $p \in Proc$, for some c, $d \in \mathbb{N}$ such that $\mu(y^p) + c \sim \mu(x^p) + d$. Now, since q_1 and q_2 are multi-timed bisimilar and by our supposition that $(q_1,\mu) \models y^p + c \sim x^p + d$ for some y^p, $x^p \in X$ for $p \in Proc$, for some c, $d \in \mathbb{N}$, we have that $\mu(y^p) + c \sim \mu(x^p) + d$.

□

As an immediate consequence of the above theorem and by instantiating the above result to the initial state of the MLTS, we obtain the following result.

proposition 20. *Given two MLTS $\mathcal{M}_1$ and $\mathcal{M}_2$, if $\mathcal{M}_1 \approx \mathcal{M}_2$ and $\mathcal{M}_1 \models \varphi$ then $\mathcal{M}_2 \models \varphi$.*

Since MTA provide a formalism for the finite description of MLTS and the clock constraints are exactly the same as those present in the syntax of the logic ML_ν, then we obtain the following result.

Theorem 61. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two MTA and $\forall\ \varphi$ formula in the logic ML_ν. If ($\mathscr{A}_1 \models \varphi$ iff $\mathscr{A}_2 \models \varphi$) then $\mathscr{A}_1 \approx \mathscr{A}_2$.*

Proof. (sketch). A proof of this theorem may be obtained from the characteristic property of TA [LLW95]. □

proposition 21. *Given two MTA $\mathscr{A}_1$ and $\mathscr{A}_2$, if $\mathscr{A}_1 \approx \mathscr{A}_2$ and $\mathscr{A}_1 \models \varphi$ then $\mathscr{A}_2 \models \varphi$.*

7.5.3 Examples of Properties

Here, we use ML_ν formulas to express multi-timed properties.

Example 29. *Consider the MTA $\mathscr{A}_q$ described in Figure 7.4 right. The initial state (q_0,μ_0) (i.e., $q_0 = (T_0, \nu_0)$) satisfies the following ML_ν formula φ:*

$$\varphi \;=\; y^q\ \underline{in}\ \exists(\,3 \geq y^q \geq 1\ \wedge\ \langle a\rangle\ true)$$

Intuitively, this formula means that the action a can be performed by the process q after a delay between 1 and 3, for instance 2 time units.

Example 30. *Consider the MTA $\mathscr{M}$ described in Figure 7.3. The initial state (q_0,μ_0) (i.e., $q_0 = (S_0, \nu_0)$) satisfies the following ML_ν formula φ:*

$$\varphi \;=\; y^q\ \underline{in}\ \exists(\,2 \geq y^q \geq 0\ \wedge\ \langle a\rangle\ S_1)\ \wedge\ (x^p \geq 1\ \wedge\ \langle a\rangle\ S_1)$$

Intuitively, this formula means that the action a can be performed by the process q after a delay between 0 and 2, for instance 1 and the action a can be performed by the process p after a delay 1 time units.

Example 31. *Consider the MTA $\mathscr{A}$ described in Figure 7.12. The state (q_1, μ_1) (i.e., $q_1 = (S_1, \nu_1)$) satisfies the following ML$_\nu$ formula φ:*

$$\begin{aligned} Z_{S_1} \;=\; & x^p\ \underline{in}\ \exists(x^p \leq 10\ \wedge\ \langle b\rangle\ Z_{S_1}) \wedge [b]\ (x^p \leq 10\ \wedge\ x^p\ \underline{in}\ Z_{S_1}) \wedge \forall\ Z_{S_1}\ \vee \\ & y^q\ \underline{in}\ \exists(y^q \leq 9\ \wedge\ \langle b\rangle\ S_0) \wedge\ x^p\ \underline{in}\ (\langle b\rangle\ S_0) \end{aligned}$$

Intuitively, this formula means that the action b can be performed by the process p before a delay 10 time units (self-loop) or the action b can be performed by the process q before a delay 9 time units.

Example 32. *The initial state (q_0, μ_0) satisfies the following ML$_\nu$ formula φ:*

$$\varphi \;=\; x^p\ \underline{in}\ \exists(x^p \geq 1\ \wedge\ \langle a\rangle\ true\ \vee\ \langle b\rangle\ true)$$

Intuitively, this formula means that the action a or b can be performed by the process p before a delay 1 time units.

Example 33. *Consider the two MTA $\mathscr{A}_p$ (left) and $\mathscr{A}_q$ (right) in Figure 7.4 with the alphabet $\Sigma = \{a\}$, the set of processes $Proc = \{p, q\}$, the set of clocks $X = \{x^p, y^q\}$ and without invariants (i.e, all the invariants are true). We could combine two MTA to a single new one, where their interactions are determined by synchronous discrete transitions and asynchronous delay transitions (all clocks of the composition evolve independently with time). For readability, we have renamed the actions in Σ to a_p and a_q. Thus, we could also combine two formulas into a single new one, where the initial state (q_0, μ_0) is the conjunction of the two initial states of $\mathscr{A}_p$ and $\mathscr{A}_q$. We could describe the multi-timed bisimulation with a single formula in ML$_\nu$. The initial state (q_0, μ_0) satisfies the following ML$_\nu$ formula φ:*

$$\begin{aligned} \varphi \;=\; & (\ [a_p]\langle a_q\rangle\ (x^p\ \underline{in}\ \exists(3 \geq x^p \geq 1\ \wedge\ \langle a_p\rangle\ true)))\ \wedge \\ & (\ [a_q]\langle a_p\rangle\ (y^q\ \underline{in}\ \exists(3 \geq y^q \geq 1\ \wedge\ \langle a_q\rangle\ true))). \end{aligned}$$

Intuitively, this formula means that the action a_p and a_q can be performed by the process p and q after a delay between 1 and 3, for instance 2 time units.

7.5.4 Model Checking

Here, we consider the model checking problem of ML$_\nu$ sentences on MTA models. This problem consists, given a ML$_\nu$ sentence φ and an MTA $\mathscr{A}$, in deciding whether the relation $\mathscr{A} \models \varphi$ holds.

Theorem 62. *The model checking problem of ML$_\nu$ on MTA is EXPTIME-complete.*

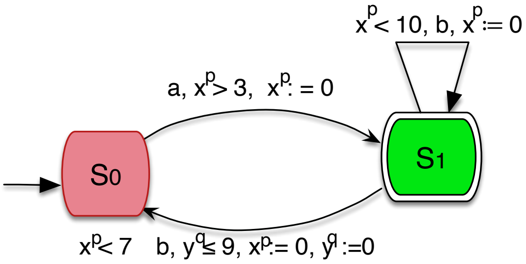

$x^p < 10$, b, $x^p := 0$
a, $x^p > 3$, $x^p := 0$
S0
S1
$x^p < 7$
b, $y^q \leq 9$, $x^p := 0$, $y^q := 0$

Figure 7.12: A Multi-timed Automata $\mathscr{A}$

Proof. EXPTIME-hardness: The proof follows from the EXPTIME-hardness of the model-checking of the logic L_ν over TA [AL99], as MTA are an extension of TA and ML_ν is the corresponding extension of L_ν: If we use a single process, $\mathsf{ML}_\nu = \mathsf{L}_\nu$ and MTA = TA. □

Proof. EXPTIME-membership: To prove EXPTIME-membership, we use the idea suggested in [LLW95]. Let $\mathscr{A}$ be a MTA, $\varphi \in \mathsf{ML}_\nu$, K the number of clocks of the automaton $\mathscr{A}$, C the maximal constant of $\mathscr{A}$ and φ, n the nesting depth of greatest fixpoint quantifier in φ. We consider the region graph $Regions(\mathscr{A},\varphi)$ [AD94] associated with $\mathscr{A}$ and the formula φ with clocks X. The region graph depends on the maximal constants with which clocks are compared in $\mathscr{A}$ and φ. Using the region graph $Regions(\mathscr{A},\varphi)$, model checking ML_ν formulas can be done in the time that is exponential in the number of K, C and n. This can be shown as in [LLW95]. Following [Alu92], $\mathscr{A} \models \varphi$ iff $\mathscr{A}' \models \varphi$, where $\mathscr{A}' = untimed(\mathscr{A})$ is the untimed automaton associated with $\mathscr{A}$ and φ (the region graph $Regions(\mathscr{A},\varphi)$). The size of $\mathscr{A}'$ is exponential in the length of the timing constraints of the given MTA automaton and in the length of the formula φ (assuming binary encoding of the constants), that is, $|\mathscr{A}'| = O((|S| + |\rightarrow_{ta}|) \cdot K! \cdot 2^K \cdot C^K)$. The region graph $\mathscr{A}'$ can be constructed in linear time, which is also bounded by $O((|S| + |\rightarrow_{ta}|) \cdot K! \cdot 2^K \cdot C^K)$ [Alu92]. On the region graph, untimed model checking can be performed in time $O((|\varphi| \cdot |\mathscr{A}'|)$. Clearly we obtain an algorithm of time complexity $O(|\varphi| \cdot (|S| + |\rightarrow_{ta}|) \cdot K! \cdot 2^K \cdot C^K)^{|\varphi|}$. □

7.5.5 Satisfiability Checking

The satisfiability checking problem, which is the dual of the model checking problem, is to check whether a given φ formula is satisfied by a multi-timed automaton $\mathscr{A}$. Formally, let φ be a ML_ν formula and $\mathscr{A}$ be a MTA, then the satisfiability problem ($\mathscr{A} \models \varphi$) is decidable. Currently, the satisfiability problem for L_ν has been shown undecidable (in fact, even for its non-recursive fragment) [JLMX14]. The satisfiability problem for the Recursive Weighted Logic (RWL) has been shown decidable by applying a variant of the region technique developed for TA [LM14a]. We could explore the possibility of using the technique presented in [LM14a] to exploit a recent decidability result [JLMX14] [LM14a].

7.6 Application of MTA and MLv

In this section, we present the experimental results related to our alternative semantics. We introduce the task graph and the characteristics of cancer problems, and we show how these problems can be modeled as a MTA. We also describe the properties of these problems in ML_{ν}.

7.6.1 Task Graph Scheduling

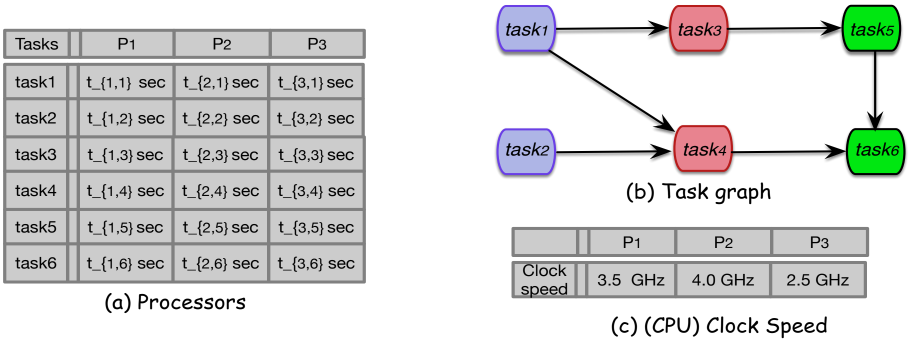
Task Graph Scheduling (TGS) is a crucial problem in distributing tasks among the processors of a (distributed) system. Since TGS is a NP-complete problem, different search methods are used to find the near-optimal schedule. In TGS, tasks are scheduled on a finite number of (homogeneous) machines or processors, while respecting some precedence constraints. Homogeneity means that all machines or processors have the same computational capabilities and that all network connections have the same physical characteristics. A task graph is a set of partially ordered tasks, with an integer number (i.e., task execution time) associated with each task. A task can be executed only if all its predecessors in this graph have completed [AKM03].

Nevertheless, for the majority of existing TGS algorithms, the task execution time is deterministic and precisely defined in advance. In fact, in the real world, assuming task execution times before execution is unrealistic, since it is not possible to obtain an accurate prediction of such execution times. However, it is more realistic to assume that task execution times may be uncertain due to the heterogeneous machines in distributed systems. Here we are interested in TGS, where tasks are scheduled on a finite number of (heterogeneous) machines or processors (heterogeneous distributed systems). Heterogeneity means that all machines or processors have different computational capabilities and all network connections have different physical characteristics. Common examples of heterogeneous systems are commodity clusters. Commodity clusters are typically upgraded with additional machines. However, these machines can be equipped with new technologies and faster computing speeds than the old machines, resulting in a heterogeneous cluster. Some tasks can be performed faster on some machines than on others. In this case, we consider an extension of the TGS described in [BFLM11] and [AKM03]. We show how MTA can be used to introduce independent clocks (clock drifts), uncertain task execution times (random values), processes, and failures.

Figure 7.13(a) shows the three processors and the execution times for six tasks. Figure 7.13(b) shows the task graph for the six tasks and the dependencies between tasks. 7.13(c) shows the CPU speed for the two processors.

7.6.2 Modeling the Task Graph Scheduling in MTA

In [AKM03] and [BFLM11], TA were used for modeling the TGS problem. These TA are composed of one automaton for each processor, one scheduler controller (automaton), and a parallel composition between them. The scheduler controller (automaton) determines when tasks are performed and on which processors. Some constraints must be respected by the graph tasks and verified by the scheduler



Tasks	P1	P2	P3
task1	t_{1,1} sec	t_{2,1} sec	t_{3,1} sec
task2	t_{1,2} sec	t_{2,2} sec	t_{3,2} sec
task3	t_{1,3} sec	t_{2,3} sec	t_{3,3} sec
task4	t_{1,4} sec	t_{2,4} sec	t_{3,4} sec
task5	t_{1,5} sec	t_{2,5} sec	t_{3,5} sec
task6	t_{1,6} sec	t_{2,6} sec	t_{3,6} sec

(a) Processors

(b) Task graph

	P1	P2	P3
Clock speed	3.5 GHz	4.0 GHz	2.5 GHz

(c) (CPU) Clock Speed

Figure 7.13: Task Graph Problem with 6 tasks

controller, such as a task cannot be scheduled until all of its predecessors have been computed. A task can be executed by any processor, but a processor cannot execute more than one task at a time. Here, we extend the TGS problem with independent clocks. Tasks may be modeled as MTA. For each task, we construct a MTA able to handle within the appropriate amount of time the requests from the tasks. Here, we model the tasks with processors, but without the scheduler controller automaton. For the $task1$ of Figure 7.13, this is as follows:

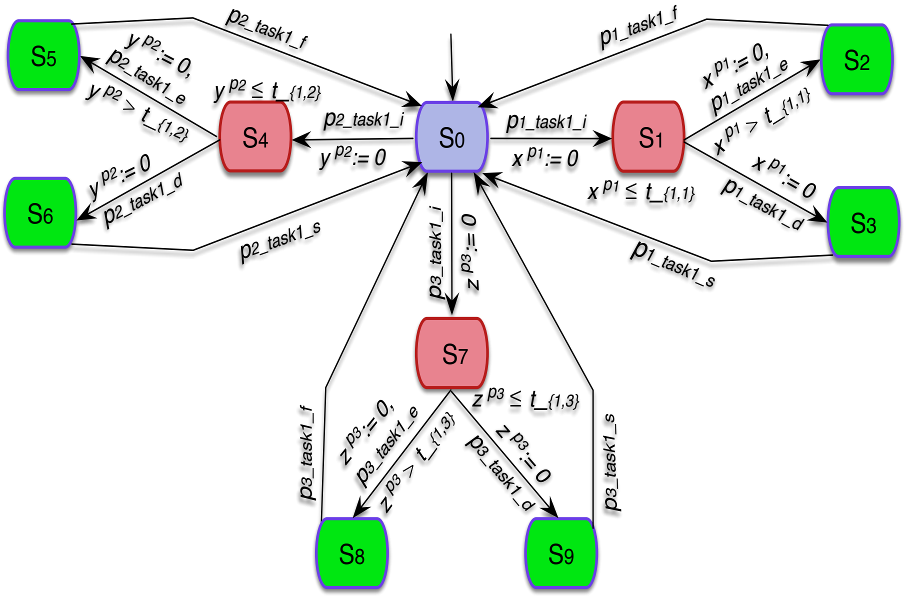


Figure 7.14: MTA model for the Task1 in figure 7.13

Figure 7.14 below shows a MTA $\mathscr{A}$ for the $task1$ of Figure 7.13 with the finite alphabet Σ ={ $p1\_task1\_i$, $p2\_task1\_i$, $p3\_task1\_i$, $p1\_task1\_e$, $p2\_task1\_e$,

$p3\_task1\_e$, $p1\_task1\_d$, $p2\_task1\_d$, $p3\_task1\_d$, $p1\_task1\_s$, $p2\_task1\_s$, $p3\_task1\_s$, $p1\_task1\_f$, $p2\_task1\_f$, $p3\_task1\_f$ }, the set of processes (processors) $Proc = \{p1, p2, p3\}$, the set of independent clocks $X = \{x^{p1}, y^{p2}, z^{p3}\}$. The actions $p1\_task1\_i$, $p2\_task1\_i$ and $p3\_task1\_i$ correspond to the start of the task (task1) either by $P1$, $P2$, or $P3$. Then, when a process (processor) is chosen (faster clock) and task1 is completed, there is a choice between moving to a location corresponding to successful completion with actions $p1\_task1\_d$, $p2\_task1\_d$ or $p3\_task1\_d$ and one to defect with actions $p1\_task1\_e$, $p2\_task1\_e$, or $p3\_task1\_e$. In both cases, we move to a location where no time can pass and immediately notify the scheduler of either the success with actions $p1\_task1\_s$, $p2\_task1\_s$, or $p3\_task1\_s$ or failure of the computation with actions $p1\_task1\_f$, $p2\_task1\_f$, $p3\_task1\_f$. The scheduler controller automaton must also change for this model since it must react to the failure signals from the processors. The independent clocks x^{p1}, y^{p2}, and z^{p3} are used to record the time that a task has been running and they are reset when a new task is started. The automata for the other 5 tasks are similar except that the action name changes and the invariants are modified to reflect the values in Figure 7.14(a).

7.6.3 Properties of the Task Graph Scheduling in ML_v

Example 34. *Consider the MTA $\mathscr{A}$ described in Figure 7.14. The initial state (q_0, μ_0) (i.e., $q_0 = (S_0, v_0)$) satisfies the following ML_v formula (property) φ:*

$$\varphi \;=\; x^{p1}\ \underline{in}\ \exists(\langle p1\_task1\_i\rangle\ S_1) \vee y^{p2}\ \underline{in}\ \exists(\langle p2\_task1\_i\rangle\ S_4) \vee \\ z^{p3}\ \underline{in}\ \exists(\langle p3\_task1\_i\rangle\ S_7)$$

Intuitively, this formula means that the action $p1\_task1\_i$ can be performed by the process $p1$ instantly (or not) and thereby to reach the S_1 location, or the action $p2\_task1\_i$ can be performed by the process $p2$ instantly (or not) and thereby to reach the S_4 location, or $p3\_task1\_i$ can be performed by the process $p3$ instantly (or not) and thereby to reach the S_7 location.

Example 35. *Consider the MTA $\mathscr{A}$ described in Figure 7.14. The state (q_1, μ_1) (i.e., $q_1 = (S_1, v_1)$) satisfies the following ML_v formula (property) φ:*

$$\varphi \;=\; x^{p1}\ \underline{in}\ \exists(x^{p1} \leq t\_\{1,1\} \wedge \langle p1\_task1\_e\rangle\ S_2) \vee x^{p1}\ \underline{in}\ \exists(\langle p1\_task1\_d\rangle\ S_3)$$

Intuitively, this formula means that the action $p1\_task1\_e$ can be performed by the process $p1$ before a delay $t\_\{1,1\}$ time units and thereby to reach the S_2 location, or the action $p1\_task1\_d$ can be performed by the process $p1$ instantly (or not) and thereby to reach the S_3 location.

7.6.4 Hallmarks of Cancer

Tumors go through certain stages and acquire certain (intermediate) phenotypic progression states (also called hallmarks), culminating in the final state of tissue invasion and metastasis. The hallmarks of cancer (or phenotypic progression states) approach is an interesting tool that summarizes the vast complexity of cancer phenotypes and genotypes into a set of discrete states. The hallmarks of cancer include six biological progression states that are acquired during the multistep development of human tumors [HW11, HW16]. The six hallmarks currently include (see Figure 7.15) Sustaining Proliferative Signaling (SPS), Evading Growth Suppressors (EGS), Resisting Cell Death (RCD, Enabling Replicative Immortality (ERI), Introducing Angiogenesis Signal (IAS), and Activating Invasion and Metastasis (AIM).

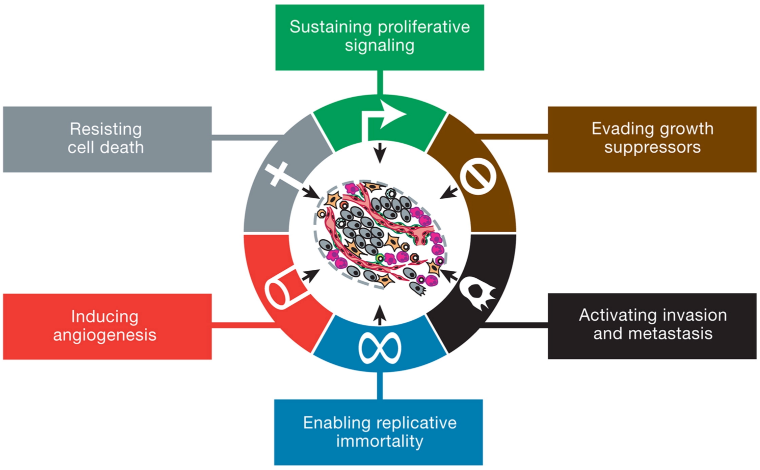

Sustaining proliferative signaling
Resisting cell death
Evading growth suppressors
Inducing angiogenesis
Activating invasion and metastasis
Enabling replicative immortality

Figure 7.15: The Hallmarks of Cancer from [HW11, HW16]

.

- Sustaining Proliferative Signaling (SPS): Normally, the body's cells need hormones and other molecules that act as alarms to tell them to grow and divide. However, cancer cells can grow without these external signals.
- Evading Growth Suppressors (EGS): Normally, cells have internal processes that control cell development and division. These processes are controlled by proteins called suppressor genes. However, cancer cells are generally resistant to growth-inhibiting signals from their neighbors.
- Resisting Cell Death (RCD): Cells can destroy themselves (also known as apoptosis). However, cancer cells lose this ability.
- Enabling Replicative Immortality(ERI): Normally, cells cannot divide indefinitely. Cells have a limited number of divisions before they become unable to divide or die. Cancer cells can divide indefinitely without senescence.

- Inducing Angiogenesis Signal (IAS): Angiogenesis is the process by which new blood vessels are formed. Cancer cells can initiate this process, which ensures that these cells have a continuous supply of oxygen and other nutrients.
- Activating Invasion and Metastasis (AIM): Cancer cells can invade surrounding tissue and spread (metastasize) to distant parts of the body.

Typically, cancer cells arise from mutations in a specific set of genes. Therefore, specific drugs are administered individually or combined in a cocktail to interfere with the functions of these genes and slow the development of cancer cells (e.g., an Avastin drug inhibits the associated signaling pathway, thus preventing growing tumors from receiving the necessary supply of oxygen and blood) [PAM+22]. The drugs that can be used in each step are listed in the Figure 7.16.

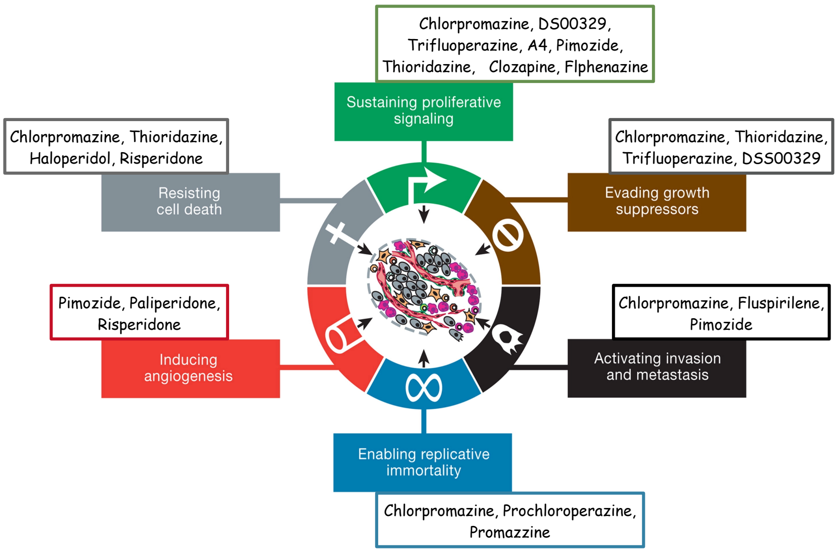

Chlorpromazine, DS00329, Trifluoperazine, A4, Pimozide, Thioridazine, Clozapine, Flphenazine
Sustaining proliferative signaling
Chlorpromazine, Thioridazine, Haloperidol, Risperidone
Resisting cell death
Chlorpromazine, Thioridazine, Trifluoperazine, DSS00329
Evading growth suppressors
Pimozide, Paliperidone, Risperidone
Inducing angiogenesis
Chlorpromazine, Fluspirilene, Pimozide
Activating invasion and metastasis
Enabling replicative immortality
Chlorpromazine, Prochloroperazine, Promazzine

Figure 7.16: The Hallmarks of Cancer represented with (cocktail) drugs from [PAM+22]

Furthermore, progressions through these hallmarks are associated with certain minimum amounts of time (i.e., depending on the patient, cancer progression in hallmarks may take days, months, or years) and drugs. Only after a certain amount of time is progression possible (i.e. moving from one state to another in the hallmarks) and a drug can be administered (drugs can stop or slow down cancer progression). It is also well known in the field of oncology that certain drugs can speed up or slow down the growth of the hallmarks of cancer.

7.6.5 Modeling the hallmarks of cancer in MTA

Here, we construct a MTA able to model the possible progression of the hallmarks of cancer (see Figure 7.17). However, this model is a simplified and generalized representation of the hallmarks of cancer.

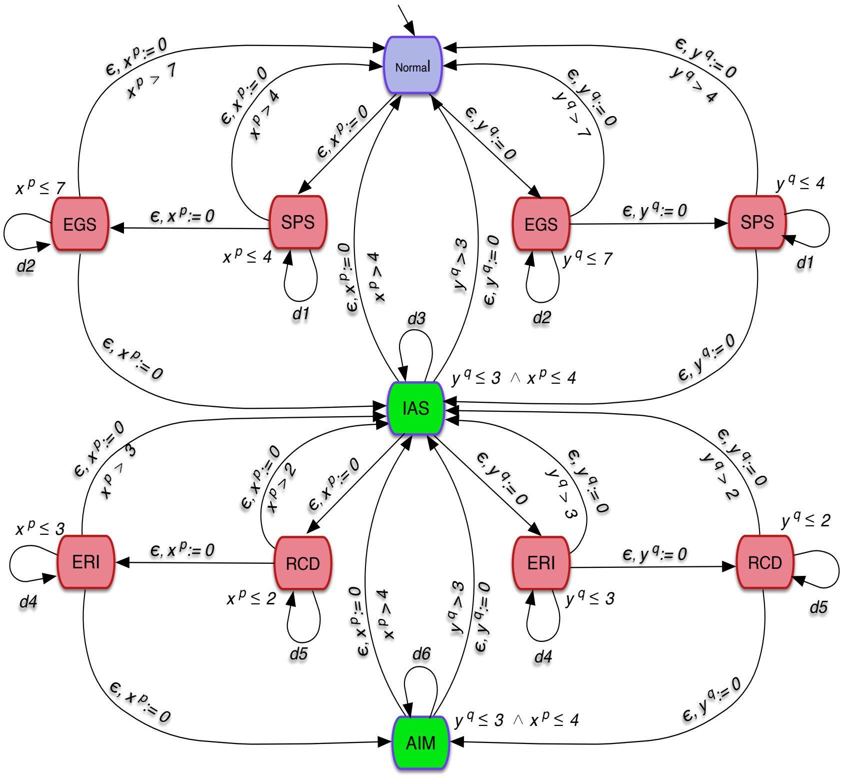


Figure 7.17: A MTA for the hallmarks of cancer from [OWM12]

Figure 7.17 below shows a MTA $\mathscr{A}$ for the hallmarks of cancer of Figure 7.16 with the finite alphabet Σ (cocktail of drugs) ={ ϵ, $d1$, $d2$, $d3$, $d4$, $d5$, $d6$ }, the set of processes (processors) $Proc$ (patients) = $\{p, q\}$, the set of independent clocks X = $\{x^p, y^q\}$. In our model, drugs can be modifying the clock rates (i.e., drugs can be speeding up the clock rates). The initial location (Normal) with the action ϵ (drug not supplied) corresponds to the start of cancer either by the patient P or q (in some patients, cancer may progress speeding up, due to different family and socio-environmental factors). Then, when a process (patient) is chosen (a faster progression of cancer), there is a choice between moving to a location SPS or EGS. In both cases, we move to a location labeled with invariant (i.e., the maximum time that the system can remain in the respective location) and action $d1$ or $d2$ (drugs). Invariants in SPS ($x^p \leq 4$) and EGS (($y^q \leq 7$) locations of the MTA $\mathscr{A}$ can force the hallmarks of cancer back to the initial location (Normal) before the transition to EGS (or SPS) becomes possible. The independent clocks x^p and y^q are used to record the time that a drug has been supplied and they are reset when a new hallmark of cancer is stated. The other locations in $\mathscr{A}$ are similar except that the action names (drugs) change (transitions) and the invariants are modified to reflect the maximum time that the hallmark of cancer can remain in the respective location.

7.6.6 Properties of the hallmarks of cancer in ML_ν

Example 36. *Consider the* $MTA\ \mathscr{A}$ *described in Figure 7.17. The initial state* (q_0, μ_0) *(i.e.,* $q_0 = (Normal, \nu_0)$*) satisfies the following* ML_ν *formula (property)* φ*:*

$$\varphi \ = \ x^p \ \underline{in}\ \exists(\langle\epsilon\rangle\ SPS) \vee y^q\ \underline{in}\ \exists(\langle\epsilon\rangle\ EGS)$$

Intuitively, this formula means that the action ϵ *can be performed by the process* p *instantaly (or not) and thereby to reach the* SPS *location, or the action* ϵ *can be performed by the process* q *instantaly (or not) and thereby to reach the* EGS *location.*

Example 37. *Consider the* $MTA\ \mathscr{A}$ *described in Figure 7.17. The state* (q_1, μ_1) *(i.e.,* $q_1 = (SPS, \nu_1)$*) satisfies the following* ML_ν *formula* φ*:*

$$\begin{aligned} Z_{SPS} \ &= (x^p \leq 4) \ \wedge\ \forall(\langle d1\rangle\ Z_{SPS}) \wedge ([d1]\ Z_{SPS}) \wedge \forall\ Z_{SPS}) \ \vee \\ & x^p\ \underline{in}\ \exists(x^p > 4\ \wedge\ (\langle\epsilon\rangle\ Normal)) \vee \exists(\ x^p\ \underline{in}\ (\langle\epsilon\rangle\ EGS)) \end{aligned}$$

Intuitively, this formula means that the action b *can be performed by the process* p *before a delay 10 time units (self-loop) or the action* b *can be performed by the process* q *before a delay 9 time units.*

7.7 Strengths and Weaknesses of the Formalisms

This section aims at showing the strengths and weaknesses of the two formalisms presented in this chapter.

7.7.1 Strengths

The main strengths are the incorporation of an alternative semantics over sequential semantics for TLTS and icTA, an extension of the classical theory of timed bisimulation with the notion of multi-timed bisimulation, and the corresponding decision algorithms: (i) a forward reachability algorithm for the parallel composition of two MTA, which will help us to minimize the state space exploration by our second algorithm, and (ii) decision algorithms for multi-timed bisimulation using the zone-based technique. The multi-timed bisimulation algorithm is EXPTIME-complete. Another strength is the definition of a formalism for specifying the timing properties of DRTS; this formalism extends L_ν by incorporating a multi-timed semantics based on MLTS with distributed clocks (ML_ν). The model checking for the extended ML_ν formula interpreted over MTA is EXPTIME-complete. The extended ML_ν logic is sound and complete. The adoption of pointwise semantics by MTA and ML_ν, provides a convenient framework for both design and implementation of reachability algorithms, multi-timed bisimulation algorithms, efficient representation and manipulation of multi-timed zones, and multi-timed zone graphs. Existing tools such as UPPAAL, and KRONOS are based on pointwise semantics.

7.7.2 Weaknesses

Since MTA are an extension of TA, then it inherits all its weaknesses of TA (TA are neither determinizable nor complementable and their inclusion problem is undecidable).

7.8 Wrap up

This chapter introduces Multi-timed Automata (MTA), which are an extension of icTA and DTA, but with an alternative semantics based on the concepts of independent clocks, multi-timed words, and Multi-timed Labeled Transition Systems (MLTS). In the second section, thanks to our alternative semantics, we were also able to extend the definition of timed bisimulation to multi-timed bisimulation. In the third section, we have shown the parallel composition between MTA. In the fourth section, we have shown an EXPTIME algorithm for deciding whether two MTA are multi-timed bisimilar. In the fifth section, we extended the timed modal L_ν logic with independent clocks. This gives us the Multi-timed modal Logic ML_ν, which we have shown to be PSPACE-complete. In the sixth section, we have shown some examples of distributed real-time models built over MTA and ML_ν. Finally, in the seventh section, we have shown the strengths and weaknesses of MTA and ML_ν.

CHAPTER 8

DISTRIBUTED CLOCKS DERIVATIVES

Independent clocks operate autonomously without any direct relationship or dependency on each other. They operate independently and maintain their own rate behavior without synchronization. The lack of a relationship between independent clocks means that changes or events in one clock do not directly affect or influence the behavior of other independent clocks. However, it is important to note that in certain scenarios, there may be indirect interactions or dependencies between independent clocks, for example, in larger systems where different components or subsystems are involved. Therefore, in this chapter, we define the formalisms of (Derivative) Multi-Timed Automata (DMTA) and (Derivative) Multi-Timed Modal Logic (DML$_v$) in the context of TT for modeling and verifying DRTS. DTMA and DML$_v$ are novel variants of TA and L$_v$ inspired by Distributed Timed Automata (DTA) and TA with Independent Clocks (icTA) [Kri99, ABG$^+$08, OLS11] which allow rate constraints on clock derivatives. Rate constraints are attached to the locations of the automaton and the clocks derivatives corresponding to the first derivative of the original ones. The clocks can advance at different rates, and the rates can be real values. Rate constraints can be expressed as a conjunction of comparisons of two clock derivatives or a clock derivative value with a natural constant of 1. If no rate constraints are specified at the locations, it is assumed that $\dot{x} = 1$ (i.e., like a TA).

Hence, we can establish a relationship between independent clocks using their clock derivatives. By using the derivatives of clocks, we can gain insights into the behavior and characteristics of the clocks themselves. The derivatives of clocks can provide valuable information about the rate or acceleration at which the clock values are changing. The magnitude of the derivative represents the rate of change, with a higher magnitude indicating a faster rate. By using the derivatives of clocks, we can compare the temporal relationships between two independent clocks or an independent clocks with 1.

In general, the reachability (and bisimulation) problem for Hybrid Automata (HA) (Regular Hybrid Automata (RHA), Stopwatch Automata(SWA), Multi-rate Timed Automata (MRTA)) is undecidable [HKPV98]. Only for a small subset of (HA) such as Initialized Rectangular Hybrid Automata (IRHA), Initialized Stopwatch Automata (ISWA), Initialized Singular Automata (ISHA), is the reachability problem known to be decidable [HKPV98]. It is known [HKPV98] that even a small generalization of MRTA or RHA can make the reachability problem undecidable.

The problem of undecidability present in SWA (i.e., a subclass of HA) arises from the possibility of stopping and resetting time (i.e., the rate at a location can be either $\dot{x} = 0$ or $\dot{x} = 1$). A SWA is like a TA with stopwatch variables instead of clocks. The problem of undecidability present in RHA and the subclass of RHA (i.e., MRTA, SHA, SWA) arises from the possibility that the continuous variables, also called flow rates (i.e., each variable is a finite-slope variable [HKPV98]), are chosen nondeterministically from a rectangular set (i.e., an interval), and that resets, guards, and invariants are defined using rectangular sets.

By inheriting the decidability conditions of TA (i.e., in In TA, the guards and invariants are limited to the conjunctions of simple integer bounds on the individual clocks, and the update is a simple assignment of the form $x = 0$) and only resetting time (i.e., $\dot{x} = 1$) of SWA, our DMTA avoids the undecidability results present in RHA and the subclass of RHA (i.e., MRTA, SHA, SWA).

This chapter is structured as follows. In section 8.1 we propose to model DRTS with Derivative Multi-timed Automata (DMTA), an extension of TA and icTA, in which rate constraints over clock derivatives are attached to the locations of the automaton. In section 8.2 we showed the parallel composition between DMTA. In section 8.3, we prove its decidability and present an EXPTIME algorithm for deciding whether two DMTA are multi-time bisimilar. In section 8.4 we propose DML$_\nu$, an extension of ML$_\nu$ with rate constraints over clock derivatives. We define the syntax and semantics of DML$_\nu$ over executions of MLTS with such semantics, and we show that its model checking problem against DML$_\nu$ is EXPTIME-complete. Section 8.5, presents some scenarios of distributed real-time systems for DMTA and DML$_\nu$. Section 8.6, presents the functionalities and details related to the implementation of our algorithms. We also present some experimental results. Finally, section 8.7, shows the strengths and weaknesses of DMTA and DML$_\nu$.

8.1 An (Derivative) Alternative Semantics for DRTS

Here we define a (derivative) alternative semantics for TA (i.e., Derivative Multi-timed Automata (DMTA)) that can be used to model DRTS. The semantics of TA is presented [AD94] with sequential (terms) semantics, such as TLTS, timed bisimulation, and timed words. In contrast, our (derivative) alternative semantics uses the rate constraints over clock derivatives. The clock derivatives denote the first derivative of the independent clocks belonging to different processes. The advantages of our (derivative) alternative semantics are (1) Our (derivative) alternative semantics can work with multi-timed words, MLTS, and rate constraints. (2) With our alternative semantics, it becomes possible to analyze the local behavior of the components *independently.* we now build the necessary notions of rate constraints and (derivative) multi-timed automata (DMTA).

8.1.1 Rate Constraints

Let $Proc$ be a non-empty set of processes. Let X be a finite set of clocks. Let X be a finite set of *clocks* and $\pi : X \to$ *Proc.* Let $\dot{X}$ be a finite set of time clock derivatives. The set $\Psi(X)$ of rate constraints over the set of clocks X is given by the following grammar:

$$\psi := true \mid \dot{x} \sim 1 \mid \dot{x} \sim \dot{y} \mid \psi_1 \wedge \psi_2$$

where $\dot{x}, \dot{y} \in \dot{X}$, $p, q \in Proc$, p =$\{x\}$, q =$\{y\}$, $x, y \in X$ and $\sim \in \{<, >, \leq, \geq, =\}$. A rate constraint ψ is a conjunction of comparisons of two clock derivative values or a clock derivative value with a natural constant 1. A classical clock (as in TA) will be described by $\dot{x} = 1$.

8.1.2 Derivative Multi-timed Automata

Inspired by TA, MTA, icTA and DTA [Kri99,ABG⁺08,AD94,OAS17], we introduce Derivative Multi-timed Automata (DMTA) to model DRTS. The semantics of a DMTA is given by our MLTS.

Definition 95 (DMTA)**.** *A DMTA is a tuple* $\mathscr{A} = (S, s_0, \Sigma, X, \to_{dmta}, Inv, R, F, \pi)$ *over Proc where :*

(i) S *is a finite set of locations,*
(ii) $s_0 \in S$ *is the initial location,*
(iii) Σ *is a finite alphabet,*
(iv) X *is a finite set of clock names,*
(v) $\to_{dmta} \subseteq S \times \Sigma \times \Phi(X) \times 2^X \times S$ *is the finite transition relation,*
(vi) $Inv : S \to \Delta(X)$ *associates to each location a clock invariant,*
(vii) $R : S \to \Psi(X)$ *associates to each location a rate constraint,*
(viii) $F \subseteq S$ *is a finite set of final locations.*
(ix) $\pi : X \to$ *Proc maps each clock to a process.*

Definition 96. *Given* $\pi : X \to$ *Proc, a clock valuation* $v : X \to \mathbb{R}_{\geq 0}$ *and* $\vec{d} \in \mathbb{R}_{\geq 0}^{Proc}$*: the valuation* $v +_{\pi} \vec{d}$ *is defined by* $(v +_{\pi} \vec{d})(x) = v(x) + \vec{d}_{\pi(x)}$ *for all* $x \in X$.

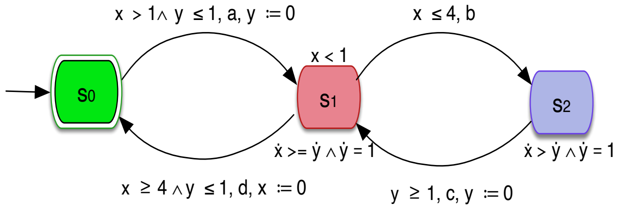


Figure 8.1: A Derivative Multi-timed Automaton $\mathscr{M}$

Definition 97. *Let $Proc$ be a non-empty set of processes. Let X be a finite set of clocks and $\pi : X \to Proc$. Given a rate constraint $\psi \in \Psi(X)$ and a tuple of functions $\tau \in Rates$, we note τ satisfies ψ at time t, as $(\tau, t) \models \psi$. In particular, the formal definition is as follows:*

$$(\tau, t) \models \dot{x} \sim 1 \Longleftrightarrow \tau_x \; is \; derivable \; at \; t \; and \; d\tau_x/dt(t) \sim 1$$

$$(\tau, t) \models \dot{x} \sim \dot{y} \Longleftrightarrow \tau_x \; is \; derivable \; at \; t \; and$$

$$\tau_y \; is \; derivable \; at \; t \; and \; d\tau_x/dt(t) \sim d\tau_y/dt(t)$$

$$(\tau, t) \models \psi_1 \; \wedge \; \psi_2 \Longleftrightarrow (\tau, t) \models \psi_1 \; and \; (\tau, t) \models \psi_2$$

$$(\tau, t) \models true \Longleftrightarrow true$$

Definition 98 (**Semantics of** DMTA)**.** *Given a DMTA $\mathscr{A} = (S, s_0, \Sigma, X, \to_{dmta}, Inv, R, F, \pi)$ and $\tau \in Rates$, the alternative semantics of $\mathscr{A}$ is given by a MLTS over X, denoted by MLTS($\mathscr{A}, \tau$) = (Q, q_0, Q_F, Σ,$\to_{mlts}$). The set of states Q consists of triples composed of a location, a clock valuation, and lastly the reference time: $Q = \{(s, \nu, t) \in S \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s)$ and $(\tau, t) \models R(s)\}$. The set of final states Q_F consists of triples $\{(s_f, \nu, t) \in F \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}_{\geq 0} \mid \nu \models Inv(s_f)$ and $(\tau, t) \models R(s_f)\}$. The starting state is $q_0 = (s_0, \nu_0, 0)$, where ν_0 is the valuation that initializes all the clocks to zero. Σ is the alphabet of $\mathscr{A}$. The transition relation $\to_{mlts}$ is defined by:*

(i) Discrete transition: A transition (q_i, a, q_{i+1}) is denoted $q_i \xrightarrow{a} q_{i+1}$ where $q_i = (s_i, \nu_i, t_i)$, $q_{i+1} = (s_{i+1}, \nu_{i+1}, t_{i+1})$, $a \in \Sigma$, there exists a transition $(s_i, a, \phi, Y, s_{i+1}) \in \to_{dmta}$, such that $\nu_i \models \phi$, $\nu_{i+1} = \nu_i[Y \leftarrow 0]$, $\nu_{i+1} \models Inv(s_{i+1})$, $(\tau, t_{i+1}) \models R(s_{i+1})$, $t_i = t_{i+1}$ and,

(ii) Delay transition: A transition $(q_i, \vec{d}, q'_i)$ is denoted $q_i \xrightarrow{\vec{d}} q'_i$ where $q_i = (s_i, \nu_i, t_i)$, $q'_i = (s_i, \nu_i + \vec{d}, t_{i+1})$, $\vec{d} = \tau(t_{i+1}) - \tau(t_i)$ and $\forall t \in [t_i, t_{i+1}] : \nu_i + (\tau(t) - \tau(t_i)) \models Inv(s_i)$ and $(\tau, t) \models R(s_i)$.

A path of $\mathscr{A}$ for $\tau \in Rates$ with its (derivative) *alternative* semantics is an initial path in MLTS($\mathscr{A}, \tau$) where discrete and continuous transitions alternate. A multi-timed word is accepted by $\mathscr{A}$ for $\tau \in Rates$ iff it is accepted by MLTS($\mathscr{A}, \tau$). The multi-timed language accepted by $\mathscr{A}$ for τ is denoted as $\mathscr{L}(\mathscr{A}, \tau)$.

Example 38. *The Figure 8.1 above shows a DMTA $\mathscr{M}$ with the finite alphabet $\Sigma = \{a, b, c, d\}$, the set of processes $Proc = \{p, q\}$, the set of independent clocks $X = \{x, y\}$, $p = \{x\}$ and $q = \{y\}$, and rates constraints $\dot{x} \geq \dot{y}$ and $\dot{y} = 1$ in location s_1, and s_2.*

8.2 Parallel Composition of DMTA

The parallel composition of two MLTS for DMTA is as 78. The parallel composition of two DMTA can be defined as:

Definition 99 (**Parallel Composition of** DMTA)**.** *Let $\mathscr{A}$ and $\mathscr{B}$ be two DMTA. The parallel composition of two DMTA $\mathscr{A}$ and $\mathscr{B}$, written $\mathscr{A} \parallel \mathscr{B}$ creates a new DMTA $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$ and $\mathscr{C} = (S_{\mathscr{C}}, s^0_{\mathscr{C}}, \Sigma_{\mathscr{C}}, X_{\mathscr{C}}, \rightarrow^{dmta}_{\mathscr{C}}, Inv_{\mathscr{C}}, R_{\mathscr{C}}, F_{\mathscr{C}}, \pi_{\mathscr{C}})$ where:*

(i) $S_{\mathscr{C}} = S_{\mathscr{A}} \times S_{\mathscr{B}}$,

(ii) $s^0_{\mathscr{C}} = (s_{\mathscr{A}}, q_{\mathscr{B}})$,

(iii) $\Sigma_{\mathscr{C}} = \Sigma_{\mathscr{A}} \cup \Sigma_{\mathscr{B}}$,

(iv) $X_{\mathscr{C}} = X_{\mathscr{A}} \cup X_{\mathscr{B}}$,

(v) $\rightarrow^{ta}_{\mathscr{C}} \subseteq S_{\mathscr{C}} \times \Sigma_{\mathscr{C}} \times \Phi(X_{\mathscr{C}}) \times 2^{X_{\mathscr{C}}} \times S_{\mathscr{C}}$ *is the transition relation given by: for a* $\in \Sigma_{\mathscr{A}} \cup \Sigma_{\mathscr{B}}$, *if* $(s_{\mathscr{A}}, a, \phi_{\mathscr{A}}, Y_{\mathscr{A}}, s'_{\mathscr{A}}) \in \rightarrow^{dmta}_{\mathscr{A}}$ *and* $(s_{\mathscr{B}}, a, \phi_{\mathscr{B}}, Y_{\mathscr{B}}, s'_{\mathscr{B}}) \in \rightarrow^{dmta}_{\mathscr{B}}$ *then* $((s_{\mathscr{A}}, s_{\mathscr{B}}), a, \phi_{\mathscr{A}} \wedge \phi_{\mathscr{B}}, Y_{\mathscr{A}} \cup Y_{\mathscr{B}}, (s'_{\mathscr{A}}, s'_{\mathscr{B}})) \in \rightarrow^{dmta}_{\mathscr{C}}$,

(vi) $Inv_{\mathscr{C}}(s_{\mathscr{A}}, s_{\mathscr{B}}) = Inv_{\mathscr{A}}(s_{\mathscr{A}}) \wedge Inv_{\mathscr{B}}(s_{\mathscr{B}})$ *for all* $s_{\mathscr{A}} \in S_{\mathscr{A}}$ *and* $s_{\mathscr{B}} \in S_{\mathscr{B}}$,

(vii) $R_{\mathscr{C}}(s_{\mathscr{A}}, s_{\mathscr{B}}) = R_{\mathscr{A}}(s_{\mathscr{A}}) \wedge R_{\mathscr{B}}(s_{\mathscr{B}})$ *for all* $s_{\mathscr{A}} \in S_{\mathscr{A}}$ *and* $s_{\mathscr{B}} \in S_{\mathscr{B}}$,

(viii) $F_{\mathscr{C}} = F_{\mathscr{A}} \times F_{\mathscr{B}}$,

(ix) $\pi_{\mathscr{C}} = \pi_{\mathscr{A}} \cup \pi_{\mathscr{B}}$.

Definition 100. *Let $\mathscr{A}$ and $\mathscr{B}$ be two DMTA. For any valuations $\nu_{\mathscr{A}}$ and $\nu_{\mathscr{B}}$ over disjoint sets of clocks $X_{\mathscr{A}}$ and $X_{\mathscr{B}}$. $X_{\mathscr{A}} \uplus X_{\mathscr{B}}$ with $\nu_{\mathscr{A}} \uplus \nu_{\mathscr{B}}$ is a valuation. Given $\nu_{\mathscr{A}\parallel\mathscr{B}}$, $\nu_{\mathscr{A}}$ and $\nu_{\mathscr{B}}$ can be denoted as $\nu_{\mathscr{A}\parallel\mathscr{B}}\rfloor_{X_{\mathscr{A}}}$ and $\nu_{\mathscr{A}\parallel\mathscr{B}}\rfloor_{X_{\mathscr{B}}}$*

Let $\mathscr{C}$ and $\mathscr{D}$ be two DMTA. Given two states $q_{\mathscr{C}} = (s_{\mathscr{C}}, \nu_{\mathscr{C}}, t_{\mathscr{C}})$ of MLTS$(\mathscr{C}, \tau)$, and $q_{\mathscr{D}} = (s_{\mathscr{D}}, \nu_{\mathscr{D}}, t_{\mathscr{D}})$ of MLTS$(\mathscr{D}, \tau)$ for any $\tau \in Rates$, the unique state of MLTS$(\mathscr{C}, \tau)$ $\parallel$ MLTS$(\mathscr{D}, \tau)$ corresponding to these states is written $((s_{\mathscr{C}}, s_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}}, t_{\mathscr{C}\parallel\mathscr{D}})$, where $\nu_{\mathscr{C}\parallel\mathscr{D}}(x) = \nu_{\mathscr{C}}(x)$ if $x \in X_{\mathscr{C}}$, and $\nu_{\mathscr{C}\parallel\mathscr{D}}(x) = \nu_{\mathscr{D}}(x)$ if $x \in X_{\mathscr{D}}$. The semantics of the parallel composition $\mathscr{C} \parallel \mathscr{D}$ will be given by means of a MLTS.

Definition 101 (**Semantics of the Parallel Composition**)**.** *Given two DMTA $\mathscr{C}$ and $\mathscr{D}$, then the MLTS generated by the parallel composition of $\mathscr{C}$ and $\mathscr{D}$ is a MLTS$(\mathscr{C}, \tau) \parallel$ MLTS$(\mathscr{D}, \tau) = (Q, Q_F, q_0, ((\Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}) \cup \mathbb{R}^X_{\geq 0}), \rightarrow_{mlts})$, where $Q = \{((s_{\mathscr{C}}, s_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}}, t_{\mathscr{C}\parallel\mathscr{D}}) \in ((S_{\mathscr{C}} \times S_{\mathscr{D}}) \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}_{\geq 0}) \mid \nu_{\mathscr{C}\parallel\mathscr{D}} \models Inv_{\mathscr{C}}(s_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s_{\mathscr{D}})$ and $\nu_{\mathscr{C}\parallel\mathscr{D}} \models R_{\mathscr{C}}(s_{\mathscr{C}}) \wedge R_{\mathscr{D}}(s_{\mathscr{D}})\}$ is the set of states. The set of final states Q_F consists of triples $\{((s^f_{\mathscr{C}}, s^f_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}}, t_{\mathscr{C}\parallel\mathscr{D}}) \in ((S^F_{\mathscr{C}} \times S^F_{\mathscr{D}}) \times \mathbb{R}^X_{\geq 0} \times \mathbb{R}_{\geq 0}) \mid \nu_{\mathscr{C}\parallel\mathscr{D}} \models Inv_{\mathscr{C}}(s^f_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s^f_{\mathscr{D}})$ and $\nu_{\mathscr{C}\parallel\mathscr{D}} \models R_{\mathscr{C}}(s^f_{\mathscr{C}}) \wedge R_{\mathscr{D}}(s^f_{\mathscr{D}})\}$. $q_0 = ((s^0_{\mathscr{C}}, s^0_{\mathscr{D}}), \nu^0_{\mathscr{C}\parallel\mathscr{D}}, 0)$, where $\nu^0_{\mathscr{C}\parallel\mathscr{D}}$ is the valuation that assigns 0 to all the clocks. Σ is the alphabet $\Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}$ and the transition relation $\rightarrow_{mlts}$ is defined by :*

(i) Discrete transition: (q, a, q') is denoted $q \xrightarrow{a} q'$ where $q = ((s_{\mathscr{C}}, s_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}}, t_{\mathscr{C}\parallel\mathscr{D}})$, $q' = ((s'_{\mathscr{C}}, s'_{\mathscr{D}}), \nu'_{\mathscr{C}\parallel\mathscr{D}}, t'_{\mathscr{C}\parallel\mathscr{D}})$, $a \in \Sigma_{\mathscr{C}} \cup \Sigma_{\mathscr{D}}$, there exists a transition $s_{\mathscr{C}} \xrightarrow{a,\phi_{\mathscr{C}},Y_{\mathscr{C}}} s'_{\mathscr{C}}$, $s_{\mathscr{D}} \xrightarrow{a,\phi_{\mathscr{D}},Y_{\mathscr{D}}} s'_{\mathscr{D}}$, such that $\nu_{\mathscr{C}\parallel\mathscr{D}} \models \phi_{\mathscr{C}} \wedge \phi_{\mathscr{D}}$, $\nu' = \nu[(Y_{\mathscr{C}} \cup Y_{\mathscr{D}}) \rightarrow 0]$, $\nu'_{\mathscr{C}\parallel\mathscr{D}} \models Inv_{\mathscr{C}}(s'_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s'_{\mathscr{D}})$, $\nu'_{\mathscr{C}\parallel\mathscr{D}} \models R_{\mathscr{C}}(s'_{\mathscr{C}}) \wedge R_{\mathscr{D}}(s'_{\mathscr{D}})$, $t' = t''$.

(ii) Delay transition: $(q, \vec{d}, q')$ is denoted $q \xrightarrow{\vec{d}} q'$ where $q = ((s_{\mathscr{C}}, s_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}}, t'_{\mathscr{C}\parallel\mathscr{D}})$, $q' = ((s_{\mathscr{C}}, s_{\mathscr{D}}), \nu_{\mathscr{C}\parallel\mathscr{D}} + \vec{d}, t''_{\mathscr{C}\parallel\mathscr{D}})$, $\vec{d} = \tau(t''_{\mathscr{C}\parallel\mathscr{D}}) - \tau(t'_{\mathscr{C}\parallel\mathscr{D}})$ and $\forall t \in [t'_{\mathscr{C}\parallel\mathscr{D}}, t''_{\mathscr{C}\parallel\mathscr{D}}]$: $\nu_{\mathscr{C}\parallel\mathscr{D}} + (\tau(t) - \tau(t'_{\mathscr{C}\parallel\mathscr{D}})) \models Inv_{\mathscr{C}}(s_{\mathscr{C}}) \wedge Inv_{\mathscr{D}}(s_{\mathscr{D}})$ and $(\tau, t) \models R_{\mathscr{C}}(s_{\mathscr{C}}) \wedge R_{\mathscr{D}}(s_{\mathscr{D}})$,

$\geq$

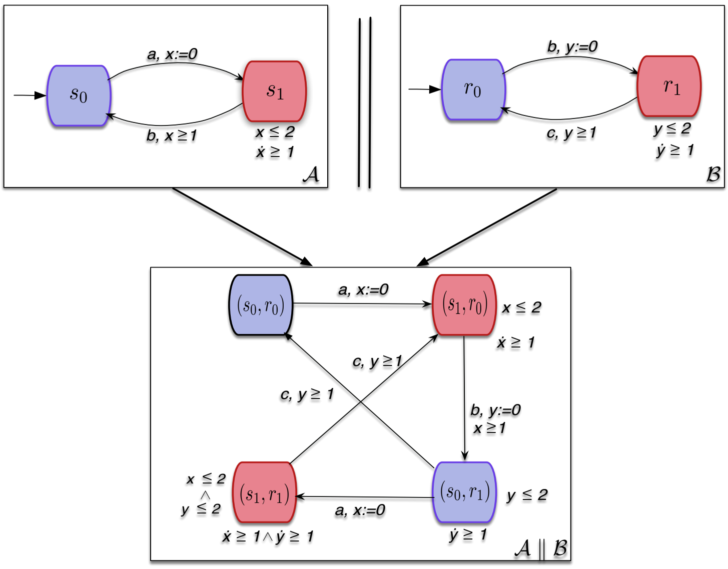


Figure 8.2: Parallel Composition of two MTA $\mathscr{A}$ and $\mathscr{B}$

Example 39. *In Figure 8.2, two DMTA $\mathscr{A}$ and $\mathscr{B}$, and their composition $\mathscr{A} \parallel \mathscr{B}$ are presented. The automaton $\mathscr{A}$ over the $Proc = \{p\}$, independent clock x and rate constraint $\dot{x} \geq 1$ in location s_1, can execute the actions a, b, with the transitions $s_0 \xrightarrow{a,true,x:=0}_{dmta} s_1$, $s_1 \xrightarrow{b,x\geq 1,\emptyset}_{dmta} s_0$. The possible transitions of the automaton $\mathscr{B}$ over the $proc$= $\{q\}$, independent clock y and rate constraint $\dot{y} \geq 1$ in location s_1, are $r_0 \xrightarrow{b,true,y:=0}_{dmta} r_1$, $r_1 \xrightarrow{c,y\geq 1,\emptyset}_{dmta} r_0$. Both components execute in parallel and synchronize through the action b. The locations of the composition (automaton) are given as pairs (s_0, r_0), (s_0, r_1) with rate constraint $\dot{x} \geq 1$, (s_1, r_0) with rate constraint $\dot{x} \geq 1$, and (s_1, r_1) with rate constraints $\dot{x} \geq 1$ and $\dot{y} \geq 1$.*

Theorem 63. *Let $\mathscr{A}$ and $\mathscr{B}$ be two DMTA, then for any $\tau \in Rates$, MLTS$(\mathscr{A}, \tau) \parallel$ MLTS$(\mathscr{B}, \tau)$ = MLTS$((\mathscr{A} \parallel \mathscr{B}), \tau)$.*

Proof. The proof consists in showing that each transition of the MLTS$(\mathscr{A}, \tau) \parallel$ MLTS$(\mathscr{B}, \tau)$ can be found in MLTS$((\mathscr{A} \parallel \mathscr{B}), \tau)$ and vice versa. Let $\mathscr{R} = \{ (((s_{\mathscr{A}}, s_{\mathscr{B}}), \nu_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}), ((s_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}}))) \mid \nu_{\mathscr{A}\parallel\mathscr{B}}(x) = \nu_{\mathscr{A}}(x) \uplus \nu_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$ and $t_{\mathscr{A}\parallel\mathscr{B}} = t_{\mathscr{A}} = t_{\mathscr{B}} \}$.

The proof follows the same lines as the proof in 58. The complete proof for this Theorem is given in Appendix C (See Theorem 68).

□

proposition 22. *Let $\mathscr{M}_1$ and $\mathscr{M}_2$ be two MLTS over the actions Σ, then $\mathscr{M}_1 \parallel \mathscr{M}_2 \approx \mathscr{M}_2 \parallel \mathscr{M}_1$.*

Proof. The proof follows the same lines as the proof in 12.

□

Definition 102 (Compositionality). *A binary relation $\approx$ between two MLTS $\mathcal{M}_1, \mathcal{M}_2$ is compositional if $\mathcal{M}_1 \approx \mathcal{M}_2$ and $\mathcal{M}_3 \approx \mathcal{M}_4$ implies $\mathcal{M}_1 \parallel \mathcal{M}_3 \approx \mathcal{M}_2 \parallel \mathcal{M}_4$.*

proposition 23. *Let $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$ be three MLTS over the set of actions Σ. For any $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}_3$, $\approx$ is compositional if and only if $\mathcal{M}_1 \approx \mathcal{M}_2 \Rightarrow \mathcal{M}_1 \parallel \mathcal{M}_3 \approx \mathcal{M}_2 \parallel \mathcal{M}_3$ (invariant under composition).*

Proof. The proof follows the same lines as the proof in 13. □

8.3 Decidability

As in MTA (see section 7.4), we present here a fundamental decidable problem used to reason about behavioral equivalence between different components of a DRTS using DMTA. Therefore, we will extend the zone-based abstraction [LLW95] to represent (derivative) multi-timed zone graphs. We show that our multi-timed bisimulation is decidable over an appropriate (derivative) multi-timed zone graph.

8.3.1 Clock Zones with Independent Clocks

To define the notion of a clock zone [LLW95] over a set of clocks X, we will consider the set $\Phi^+(X)$ of all the diagonal constraints over X. But because our clocks evolve at rates that can be independent of each other, and because there are rate constraints associated with locations, we need to adjust certain operations on clock zones. This is described by rate constraints.

Example 40. *Consider a clock zone $\mathcal{Z} = \{(4 \leq x_1 \leq 7) \wedge (2 \leq x_2 \leq 5)\}$, rate constraint $\psi = (\dot{x}_2 = 1) \wedge (\dot{x}_1 \geq \dot{x}_2)$, and clock invariant $Inv = \{(x_1 \leq 6) \wedge (x_2 \geq 4)\}$. Figure 8.3 presents an example of a) a clock zone $\mathcal{Z}$, b) the derivative lines (rate cone), c) the time successor of the clock zone $\mathcal{Z}$ ($\mathcal{Z} \uparrow_\psi$). The time successor of $\mathcal{Z}$ is: $\mathcal{Z} \uparrow_\psi = \{(x_1 \geq 4) \wedge (1 \leq x_2 \leq 5) \wedge (x_2 - x_1 \leq 1)\}$. Figure 8.4 presents an example of the intersection of the clock zone $\mathcal{Z} \uparrow$ (Figure 8.3) c) with the invariant $Inv = \{(x_1 \leq 6) \wedge (x_2 \geq 4)\}$. Figure 8.5 presents an example of a) a clock zone $\mathcal{Z}$, b) the derivative lines (rate cone), c) the time predecessor of the clock zone $\mathcal{Z}$ ($\mathcal{Z} \downarrow_\psi$). The time predecessor of $\mathcal{Z}$ is: $\mathcal{Z} \downarrow_\psi = \{(x_1 \leq 7) \wedge (1 \leq x_2 \leq 5) \wedge (x_2 - x_1 \leq 1)\}$.*

8.3.2 Operations on Clock Zones with Independent Clocks

Here, we will extend the semantics of some operations on clock zones to their multi-timed version (time successor and predecessor). The intersection, restricting projection, clock reset, and inverse clock reset operations retain the same semantics as in section 5.3.4.1. We note that clock zones are always convex. To implement our decidable algorithms, we need to be able to compute successors and predecessors of zones for delay and action transitions of DMTA.

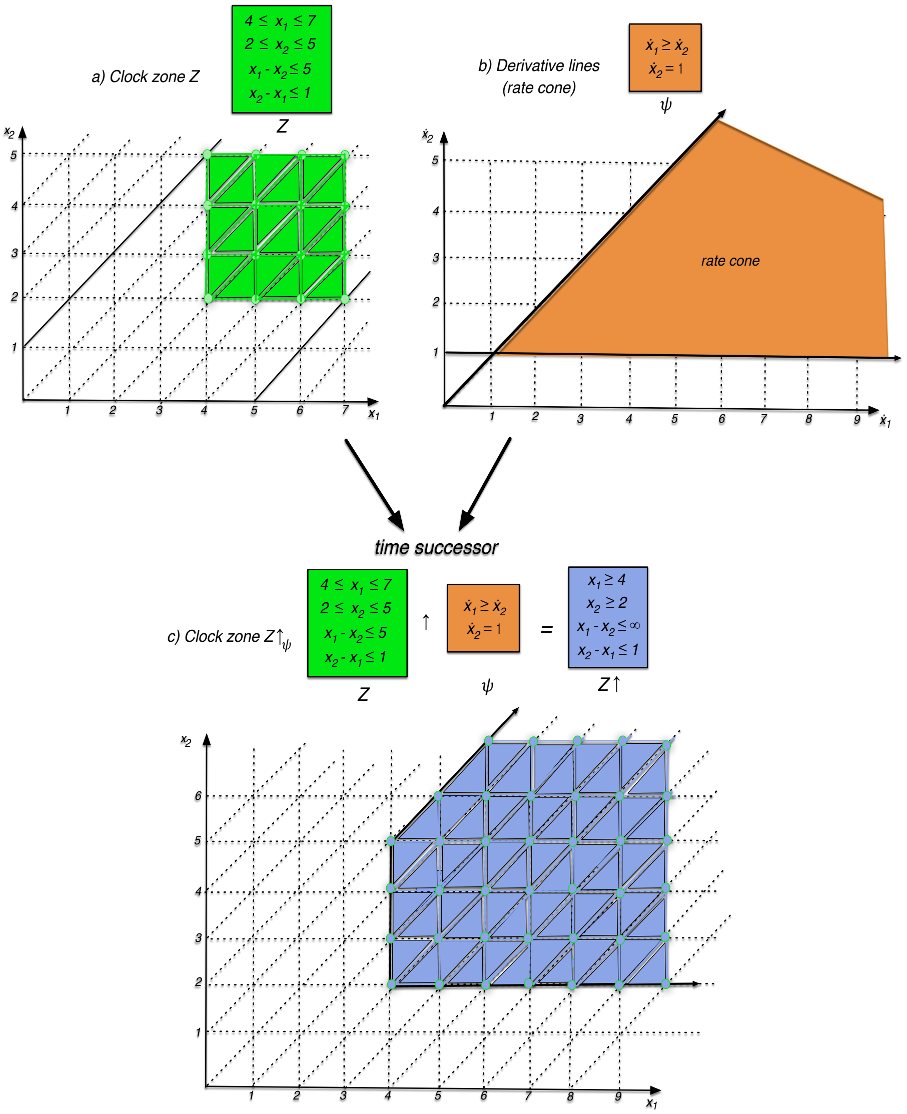


Figure 8.3: a) The clock zone $\mathcal{Z}$, b) Derivative lines (rate cone), c) The time successor of $\mathcal{Z}$ ($\mathcal{Z}\uparrow_{\psi}$)

Definition 103 (**Semantic Operations on clock zones**). *Let $\mathcal{Z}$ be a clock zones. The semantics of the time successor and time predecessor on a clock zone can be defined:*

(i) *Time successor:* $\mathcal{Z}\uparrow_{\psi} = \{v + \vec{d} \mid v \in \mathcal{Z},\ \vec{d} \in \mathbb{R}^{Proc}_{\geq 0}, \exists t, t' > 0, t \leq t',\ and\ \exists \tau \in Rates, \vec{d} = \tau(t') - \tau(t)\ and\ \exists \psi \in \Psi(X),\ (\tau, t) \models \psi\}$,

(ii) *Time predecessor:* $\mathcal{Z}\downarrow_{\psi} = \{v - \vec{d} \mid v \in \mathcal{Z},\ \vec{d} \in \mathbb{R}^{Proc}_{\geq 0}, \exists t, t' > 0, t \leq t',\ and\ \exists \tau \in Rates, \vec{d} = \tau(t') - \tau(t)\ and\ \exists \psi \in \Psi(X),\ (\tau, t) \models \psi\}$.

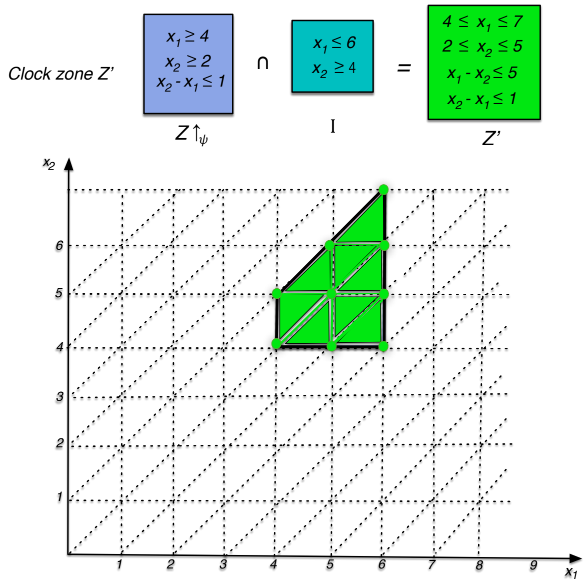


Figure 8.4: The clock zone $\mathcal{Z}'$ after intersection with the invariant Inv= $\{x_1 \leq 6 \wedge x_2 \geq 4\}$

proposition 24. *Let $\mathcal{Z}$ be a clock zones. Then, $\mathcal{Z}\uparrow_{\psi}$ and $\mathcal{Z}\downarrow_{\psi}$ are also clock zones.*

Proof. Let $\mathcal{Z}$ be zones, then we need to prove the following operations are clock zones:

(i) $\mathcal{Z}\uparrow_{\psi}$,
(ii) $\mathcal{Z}\downarrow_{\psi}$,

The proof follows the same lines as the proof in $14(v)$. The complete proof for this Theorem is given in Appendix C (See Propostion 31).

□

8.3.3 Symbolic Representation Structures

In [Dil90] [BY04] the authors propose a data structure (DBM) for representing clock zones and manipulating them. The DBM representation is the most common data structure used for reachability analysis and provides a technique for computing representations of their sets of reachable configurations. Here we will use the same data structures (DBMs) [Dil90] [BY04]. However, in our DBMCs (Difference Bound Matrices with Independent Clocks) representation, we use a set of independent clocks, which implies that differences of clocks can change over time, since all clocks can evolve at different rates, instead of differences of clocks as in classical DBMs, where all clocks evolve at the same time. Thus, each of our DBMCs is associated with a set of rate constraints over the set of independent clocks. The rate constraints associated with each of our DBMCs are defined from a finite set of predefined comparisons involving two clock derivative values or one clock derivative value

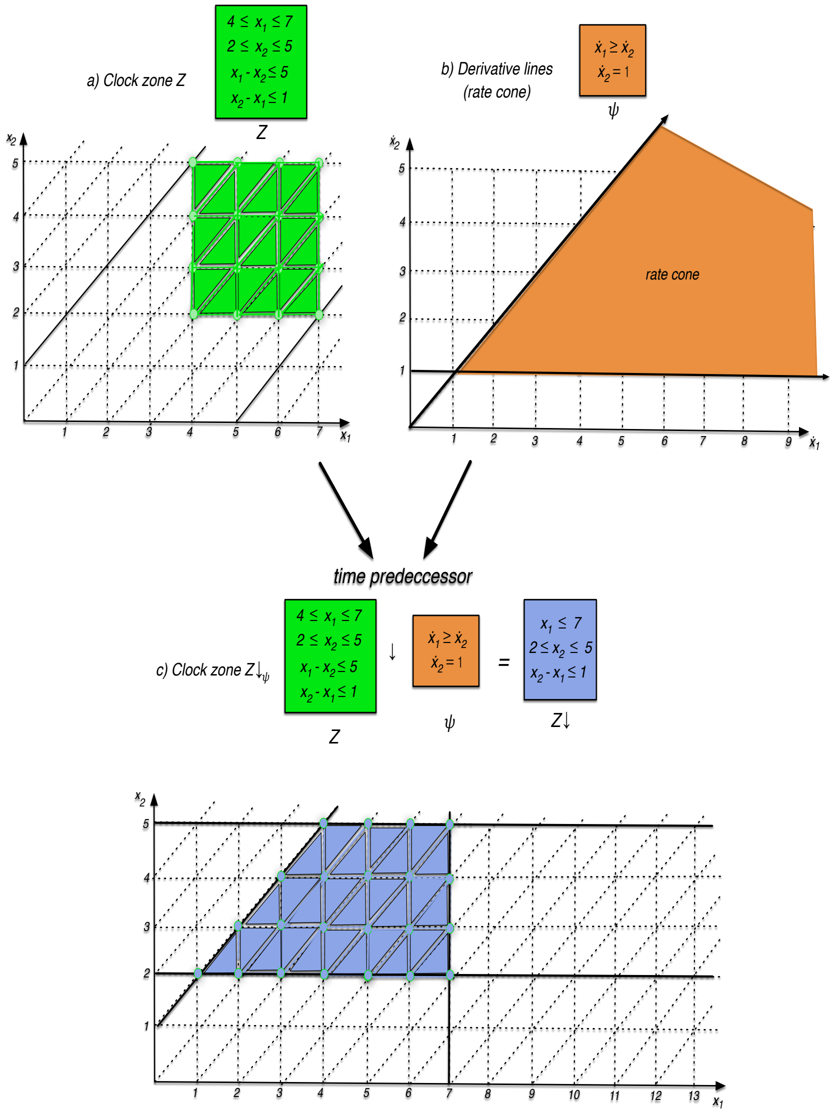


Figure 8.5: a) The clock zone $\mathcal{Z}$, b) Derivative lines (rate cone), c) The time predecessor of $\mathcal{Z}$ ($\mathcal{Z} \downarrow_\psi$)

with a natural constant of 1. In the following, the definition of our DBM and the processing methods for this structure are presented.

8.3.3.1 Difference Bound Matrices with Independent Clocks

In single-timed semantics, a DBM is a $(n+1) \times (n+1)$ matrix where an entry (i, j) is the upper bound of the clock constraint $x_i - x_j$, represented as $x_i - x_j \le d_{i,j}$ or

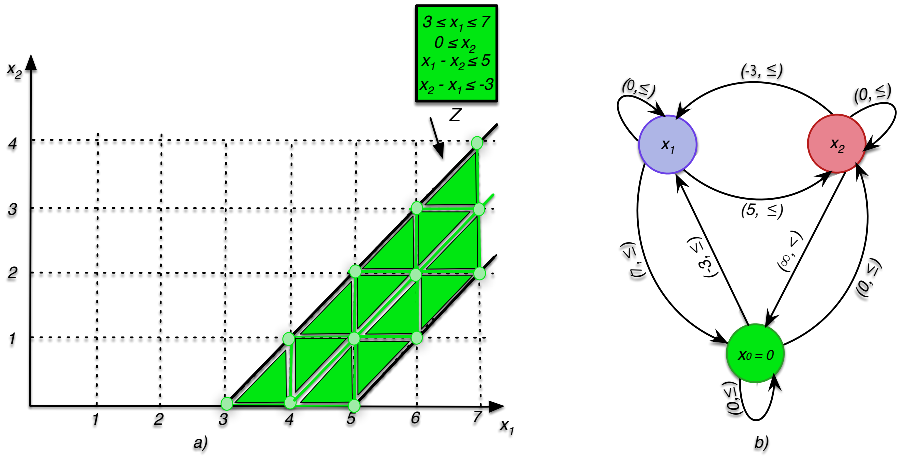


Figure 8.6: a) A clock zone $\mathscr{Z}$ and b) Directed weighted graph

$x_i - x_j < d_{i,j}$. Here, instead of using a difference of perfectly synchronized clocks in the entries, we will use independent (local) clocks (i.e., because our clocks evolve at rates that can be independent of each other). Our DBMs and their canonical form is defined analogously to standard DBMs (see definition 44 and 45). For example, consider the following clock zone with independent (local) clocks:

$$\mathscr{Z} = [\![3 \leq x_1 \leq 7 \wedge x_2 \geq 0 \wedge 3 \leq x_1 - x_2 \leq 5]\!]$$

That clock zone can be represented by the matrix $\mathscr{D}$:

$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \mathscr{D} = \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} (0,\leq) & (-3,\leq) & (0,\leq) \\ (7,\leq) & (0,\leq) & (5,\leq) \\ (\infty,<) & (-3,\leq) & (0,\leq) \end{pmatrix} \end{array}$$

In Figure 8.6(a) we can see the representation of a clock zone $\mathscr{Z}$ and Figure 8.6(b) we can see the directed weighted graph for the clock zone $\mathscr{Z} = [\![3 \leq x_1 \leq 7 \wedge x_2 \geq 0 \wedge 3 \leq x_1 - x_2 \leq 5]\!]$. Observe that this $\mathscr{D}$ does not capture many implied constraints. For instance, $\mathbf{d}_{2,0} = (\infty,<)$ which implies $x_2 < \infty$ but since $x_1 \leq 7$ and $x_2 - x_1 \leq -3$, this bound can be further tightened (canonicalization). Performing canonicalization to $\mathscr{D}$, we can get the following:

$$\begin{array}{cc} & \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \mathscr{D} = \begin{array}{c} x_0 \\ x_1 \\ x_2 \end{array} & \begin{pmatrix} (0,\leq) & (-3,\leq) & (0,\leq) \\ (7,\leq) & (0,\leq) & (5,\leq) \\ (4,\leq) & (-3,\leq) & (0,\leq) \end{pmatrix} \end{array}$$

In the case of our DBMs with independent clocks, it is possible to decide the relation between weights and nodes using classical operations on zones, as in the case of standard DBMs. However, we must first define a new data structure for symbolic representation of rate constraints, called DBM with clock derivatives (DBMCs). Our new symbolic data structure is defined in an analogous way to standard DBMs, but instead of using the difference of the clocks in the entries, we will use the difference of the clocks derivatives in the entries. We will use our DBMC first to check the consistency of the conjunction of atomic rate constraints using a consistent algorithm and to check if the conjunction of atomic rate constraints is satisfied. Second, we will use the difference of clock derivatives computed and present in the entries of the DBMC to modify the difference of clocks computed and present in the entries of the DBMC, especially in the time successor and time predecessor. A DBMC is thus just a normal DBM, but has the role of expressing rate constraints. In the following, the definition of DBMCs and processing methods for this structure are presented.

Definition 104 (**Difference Bound Matrix with Derivative Clocks**). *A Difference Bound Matrix (DBMC) over the set of n derivatives clocks* $\{\dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n\}$ *is a* $(n+1) \times (n+1)$ *square matrix (arrowhead matrix) of* $(\{-1,0,1\} \times \{<, \leq\}) \cup \{(\infty, <)\}$ *with rows and columns indexed by* $\{\dot{x}_0, \dot{x}_1, \dot{x}_2, \ldots, \dot{x}_n\}$*. The DBMC can be represented as follows:*

$$\mathscr{E} = (\boldsymbol{e}_{i,j})_{0 \leq i,j \leq n}$$

where each $\mathbf{e}_{i,j}$ *is of the form* $(e_{i,j}, \preceq)$*,* $\preceq \in \{<, \leq\}$ *and* $e_{i,j} \in \{-1, 0, 1\}$ *is called a bound. Formally, the semantics of DBMC* $\mathscr{E}$ *is the rate constraints:*

$$\psi = (\dot{x}_0 = 0) \;\wedge \bigwedge_{0 \leq i \neq j \leq n}^{n} \dot{x}_i - \dot{x}_j \preceq e_{i,j}$$

where $\dot{x}_i$ *is a clock labelling row i,* $\dot{x}_j$ *is a clock labelling the column j and the clock* $\dot{x}_0$ *is always equal to 0.*

Since the variable x_0 is always equal to 0, it can be used for expressing rate constraints that only involve a single variable (clock derivative). Actually, the language of rate constraints only allows comparing with 1. Thus, $\mathbf{e}_{i,0} = (e_{i,0}, \preceq)$ means $\dot{x}_i \preceq e_{i,0}$ and $e_{i,0} = 1$ (or $e_{i,0} = \infty$). Similarly, $\mathbf{e}_{0,j} = (e_{0,j}, \preceq)$ means $-\dot{x}_j \preceq e_{0,j}$. Also, for each clock difference $\dot{x}_i - \dot{x}_i$, let $\mathbf{e}_{i,i} = (0, \leq)$ or for each unbounded clock difference $\dot{x}_i - \dot{x}_j$ with $i \neq j$ and $i, j \geq 1$, let $\mathbf{e}_{i,j} = (0, \leq)$ (or $e_{i,j} = \infty$). A DBMC $\mathscr{E}$ has always satisfied the definition of rates, which implies that $\dot{x}_i > 0$ for $i \leq i \leq n$, thus $\mathscr{E}_{0,1}$ is at least $(0, <)$. For instance, consider again the clock zone $\mathscr{Z} = [[3 \leq x_1 \leq 7 \wedge 0 \leq x_2 \leq 4 \wedge 3 \leq x_1 - x_2 \leq 5]]$, which can be presented by matrix $\mathscr{D}$ and the conjunction of atomic rate constraints $\psi = \{\dot{x}_2 \geq 1 \wedge \dot{x}_2 \geq \dot{x}_1\}$, which can be presented by matrix $\mathscr{E}$:

$$\mathscr{D} = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \!\!\begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0,\leq) & (-3,\leq) & (0,\leq) \\ (7,\leq) & (0,\leq) & (5,\leq) \\ (4,\leq) & (-3,\leq) & (0,\leq) \end{pmatrix} \end{array}, \quad \mathscr{E} = \begin{array}{c} \\ \dot{x}_0 \\ \dot{x}_1 \\ \dot{x}_2 \end{array} \!\!\begin{array}{c} \begin{array}{ccc} \dot{x}_0 & \dot{x}_1 & \dot{x}_2 \end{array} \\ \begin{pmatrix} (0,\leq) & (\infty,<) & (-1,\leq) \\ (\infty,<) & (0,\leq) & (0,\leq) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix} \end{array}$$

The canonical form for DBMCs is the same as for standard DBMs. The canonical form algorithm for DBMCs follows the same principles as the classical Floyd-Warschall shortest path algorithm [Dil90] for the single-timed case. We also need

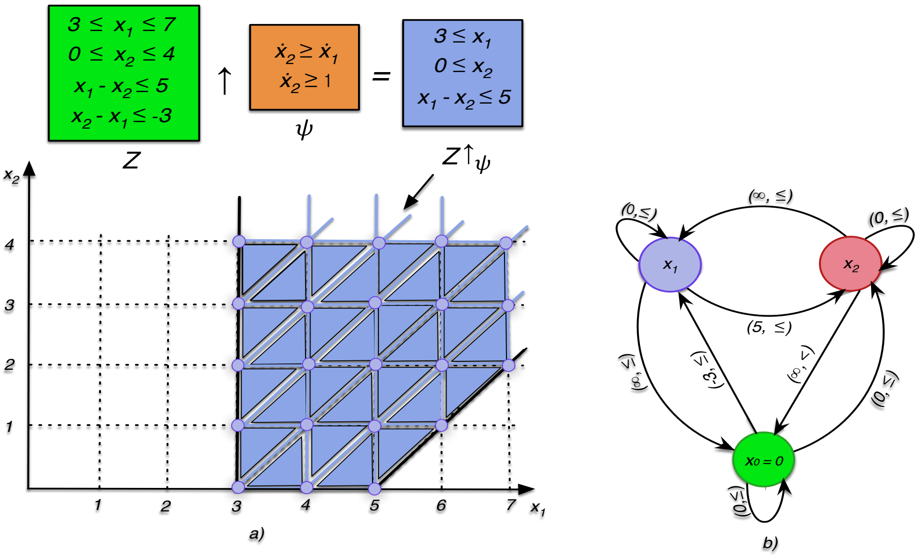


Figure 8.7: a) A clock zone $\mathscr{Z}$ and b) Directed weighted graph

to check the consistency of the conjunction of atomic rate constraints and whether the conjunction of atomic rate constraints is satisfied. To check the consistency and satisfiability of our DBMCs, the following operations are necessary: checking the standard emptiness of the derivative lines (rate cones) represented by DBMC and checking whether the derivative lines (rate cones) represented by the DBMC satisfy a given set of rate constraints. In addition, the following operations transforming the clock zones thanks to the derivative lines (rate cone) represented by the DBMC are necessary: time successor and time predecessor.

Thus, the canonical DBM $\mathscr{D}$ (see above) representing clock zone $\mathscr{Z} = [[3 \leq x_1 \leq 7 \wedge 0 \leq x_2 \leq 4 \wedge 3 \leq x_1 - x_2 \leq 5]]$ and the DBMC $\mathscr{E}$ (see above) representing the conjunction of atomic rate constraints $\psi = \{\dot{x_2} \geq 1 \wedge \dot{x_2} \geq \dot{x_1}\}$, the operation time successor returns a DBM $\mathscr{D}'$ (see below) that represents clock zone $\mathscr{Z} \uparrow_\psi$, i.e. all valuations that can be reached by valuations in $\mathscr{Z}$ with delay following a rate that satisfies ψ. The time successor $\mathscr{Z} \uparrow_\psi$ is computed by removing the upper bounds of all individual clocks, which is done by replacing all entries in the first column of $\mathscr{D}$ ($\mathbf{d}_{i,0}$ and $1 \leq i \leq n$) by $(\infty, <)$ and, because all clocks evolve at rates that can be independent of each other, the upper bounds of the constraints on the differences between clocks could be also removed, which is done by replacing the entries in the diagonal clock constraints of $\mathscr{D}$ ($\mathbf{d}_{i,j}$ and with $i \neq j$ and $i, j \geq 1$) by $(\infty, \leq)$, when the entries $\mathbf{e}_{i,j} = (\infty, <)$ with $i \neq j$ and $i, j \geq 1$).

In Figure 8.7, we assume that the matrices $\mathscr{D}$ and $\mathscr{E}$ are canonical, and they represent non-empty clock zone and non-empty derivative lines (rate cone). The canonical form algorithm and check emptiness algorithm used by ours DBM and DBMC processing are based on those described in [BY04] [AD94]. In Figure 8.7(a) we can see the representation of a clock zone $\mathscr{Z}$ in conjunction with the rate constraints

ψ. Figure 8.7(b) we can see the directed weighted graph for the clock zone $\mathcal{Z}$ = $[[3 \leq x_1 \leq 7 \wedge 0 \leq x_2 \leq 4 \wedge 3 \leq x_1 - x_2 \leq 5]]$ and rate constraints ψ= $\{\dot{x_2} \geq 1 \wedge \dot{x_2} \geq \dot{x_1}\}$:

$$\mathcal{D}' = \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \end{array} \begin{array}{c} \begin{array}{ccc} x_0 & x_1 & x_2 \end{array} \\ \begin{pmatrix} (0,\leq) & (-3,\leq) & (0,\leq) \\ (\infty,<) & (0,\leq) & (5,\leq) \\ (\infty,<) & (\infty,<) & (0,\leq) \end{pmatrix} \end{array}$$

8.3.3.2 Operations on DBM with Independent Clocks

We need basically to implement two operations on our DBMs: time successor, time predecessor.

Time Successor

As in DBMs, it is possible to define the time successor operation of a canonical DBM $\mathcal{D}$ concerning a set of rate constraints ψ, which can be presented by a canonical DBMC $\mathcal{E}$. Represented by two canonical matrices $\mathcal{D}$ and $\mathcal{E}$ where $\mathcal{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and $\mathcal{E} = (\mathbf{e}_{i,j})_{0\leq i,j\leq n}$. Computing the time successor operation of a DBM $\mathcal{D}'$ where $\mathcal{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j<n}$, consists first in removing of $\mathcal{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ all the upper bounds on the values of clocks, that is, for each $0 \leq i \leq n$, $\mathbf{d}'_{i,0} = (\infty,<)$. Second, the upper bounds of the constraints on the differences between clocks are removed, which is done by replacing the entries in the diagonal clock constraints of $\mathcal{D}'$ ($\mathbf{d}'_{i,j}$ and with $i \neq j$ and $i, j \geq 1$) by $(\infty,<)$, when the entries $\mathbf{e}_{i,j} = (\infty,<)$ with $i \neq j$ and $i, j \geq 1$. Algorithm 8.1 describes the time successor operation of a DBM. The algorithm works as follows: it repeatedly removes the upper bounds of all individual clocks, which is done by replacing all elements in the first column of $\mathcal{D}'$ by $(\infty,<)$ and replacing the diagonal elements of $\mathcal{D}'$ ($\mathbf{d}'_{i,j}$ and with $i \neq j$ and $i, j \geq 1$) by $(\infty,<)$ when the entry $\mathbf{e}_{i,j} = (\infty,<)$ with $i \neq j$ and $i, j \geq 1$ (for the strict successor, we can replace the lower bounds by their strict version).

Time Predecessor

As in DBMs, it is possible to define the time predecessor operation of a canonical DBM $\mathcal{D}$ concerning a set of rate constraints ψ, which can be presented by a canonical DBMC $\mathcal{E}$. Represented by two canonical matrices $\mathcal{D}$ and $\mathcal{E}$ where $\mathcal{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and $\mathcal{E} = (\mathbf{e}_{i,j})_{0\leq i,j\leq n}$. Computing the time predecessor operation of a DBM $\mathcal{D}'$ where $\mathcal{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j<n}$, consists first in removing of $\mathcal{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ all the lower bounds on the values of clocks, that is, for each $1 \leq i \leq n$, $\mathbf{d}'_{0,i} = (0,\leq)$. Second, the lower bounds of the constraints on the differences between clocks are removed, which is done by replacing the entries in the diagonal clock constraints of $\mathcal{D}'$ ($\mathbf{d}'_{i,j}$ and with $i \neq j$ and $i, j \geq 1$) by $(0,<)$, when the entries $\mathbf{e}_{i,j} = (0,<)$ with $i \neq j$ and $i, j \geq 1$. Algorithm 8.2 describes the time successor operation of a DBM. The algorithm works as follows: it repeatedly removes the lower bounds of all individual clocks, which is done by replacing all elements in the first column of $\mathcal{D}'$ by $(0,<)$ and replacing the diagonal elements $\mathbf{d}'_{i,j}$ and with $i \neq j$ and $i, j \geq 1$ by $(0,<)$ when the entry $\mathbf{e}_{i,j}$

```
Input: A $\mathscr{D} = (\mathbf{d}_{i,j})_{0\le i,j\le n}$ and a $\mathscr{E} = (\mathbf{e}_{i,j})_{0\le i,j\le n}$
Output: A $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\le i,j\le n}$
int i, j;
DBM $\mathscr{D}'$;
DBM TimeSuccOperator(DBM $\mathscr{D}$, DBMC $\mathscr{E}$){
    for(i=0; i<$\mathscr{D}$.size; i++){
       $\mathscr{D}'$[i,0] = $(\infty,<)$;
    }
    for(i=1; i<$\mathscr{D}$.size; i++){
      for(j=1; j<$\mathscr{D}$.size; j++){
      if( i != j && ($\mathscr{E}$[i,j]== $(\infty,<)$ && $\mathscr{E}$[j,i]== $(0,\le)$){
        $\mathscr{D}'$[i,j] = $(\infty,<)$;
       }
      else if( i != j && ($\mathscr{E}$[j,i]== $(\infty,<)$ && $\mathscr{E}$[i,j]== $(0,\le)$){
        $\mathscr{D}'$[j,i] = $(\infty,<)$;
      }
      else continue;
    }
  }
return $\mathscr{D}'$;
}
```

Algorithm 8.1: Time Successor Algorithm for independent clocks DBM.

$= (0,<)$ with $i \neq j$ and $i, j \geq 1$ (for the strict predecessor, we can replace the upper bounds by their strict version).

8.3.4 (Derivative) Multi-timed Zone Graphs

Like TA, DMTA cannot be analyzed by finite-state techniques, since its associated MLTS has infinitely many states. Therefore, it must be analyzed symbolically. Here we define the multi-timed zone graph using the independent local clocks and rate constraints, and we extend the well-known zone graph for TA [AD94] [LLW95]. More precisely, we extend the symbolic discrete successor and predecessor operations on clock zones for a DMTA $\mathscr{A}$. These operations on clock zones can be efficiently implemented on DBM [Dil90] [BY04]. The symbolic discrete successor and predecessor operations on (derivative) clock zones follow the same lines as in Section 7.4.2. The complete definitions are given in the appendix C (see Definitions 116 and 117).

proposition 25. *Let $\mathscr{A}$ be a DMTA, $e = (s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of a DMTA $\mathscr{A}$ and $(s, \mathcal{Z})$ be a zone, then $post(\mathcal{Z}, e) \uparrow_{\psi} = ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_X \cap Inv(s'))$ and $pred(\mathcal{Z}', e) \downarrow_{\psi} = ((\mathcal{Z}' \uparrow_X \cap \phi) \cap Inv(s))$.*

Proof. Let $(s, \mathcal{Z})$ be a zone and $e = (s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of an DMTA $\mathscr{A}$, then we need to prove the following equalities:

```
Input: A $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and a $\mathscr{E} = (\mathbf{e}_{i,j})_{0\leq i,j\leq n}$
Output: A $\mathscr{D}' = (\mathbf{d}'_{i,j})_{0\leq i,j\leq n}$
int i, j;
DBM $\mathscr{D}'$;
DBM TimePredecOperator(DBM $\mathscr{D}$, DBMC $\mathscr{E}$){
    for(i=0; i<$\mathscr{D}$.size; i++){
       $\mathscr{D}'$[0,i] = $(0,\leq)$;
    }
    for(i=1; i<$\mathscr{D}$.size; i++){
      for(j=1; j<$\mathscr{D}$.size; j++){
      if( i != j && ($\mathscr{E}$[i,j]== $(0,<)$ && $\mathscr{E}$[j,i]== $(\infty,\leq)$){
        $\mathscr{D}'$[i,j] = $(0,<)$;
       }
      else if( i != j && ($\mathscr{E}$[j,i]== $(0,<)$ && $\mathscr{E}$[i,j]== $(\infty,\leq)$){
        $\mathscr{D}'$[j,i] = $(0,<)$;
      }
      else continue;
    }
  }
return $\mathscr{D}'$;
}
```

Algorithm 8.2: Time Predecessor Algorithm for independent clocks DBM.

(i) $\mathsf{post}(\mathcal{Z}, e)\uparrow_{\psi} = ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_Y \cap Inv(s'))$,
(ii) $\mathsf{pred}(\mathcal{Z}', e)\downarrow_{\psi} = ((\mathcal{Z}' \uparrow_Y \cap \phi) \cap Inv(s))$.

The proof follows the same lines as the proof in 15. The complete proof for this proposition is given in Appendix C (See Propostion 32).

□

A symbolic semantics of DMTA called (derivative) multi-timed zone graph is defined as follows:

Definition 105 (**(Derivative) Multi-timed Zone Graph**). *Given a DMTA $\mathscr{A}$, its infinite multi-timed zone graph (DMZG($\mathscr{A}$)) is a transition system DMZG($\mathscr{A}$) = $(Q, q_0, (\Sigma \cup \{\epsilon\}), \rightarrow_{ZG})$, where:*

(i) Q consists of pairs $q = (s, \mathcal{Z})$ where $s \in S$, and $\mathcal{Z} \in \Phi^+(X)$ is a non-empty clock zone with $\mathcal{Z} \subseteq Inv(s)$,
(ii) $q_0 \in Q$ is the initial zone $q_0 = (s_0, \mathcal{Z}_0)$ with $\mathcal{Z}_0 = [\![\bigwedge_{x\in X} x = 0]\!]$,
(iii) Σ is the set of labels of $\mathscr{A}$,
(iv) $\rightarrow_{DMZG} \subseteq Q \times (\rightarrow_{dmta} \cup \{\epsilon\}) \times Q$ is a set of transitions, where each transition in DMZG($\mathscr{A}$) is a labeled by a transition $e = (s, a, \phi, Y, s') \in \rightarrow_{dmta}$, where s and s' are the source and target locations, ϕ is a clock constraint defining the guard of the transition, a is the action of the edge and Y is the set of clocks to be reset by

the transition in the DMTA $\mathscr{A}$. For each $e \in \Sigma$, transitions are defined by the rules:

(a) *For every $e = (s, a, \phi, Y, s')$ in $\mathscr{A}$ and zone $(s, \mathcal{Z})$ already in Q, there exists a discrete transition (q, e, q'), where $q = (s, \mathcal{Z}) \xrightarrow{e}_{ZG} q' = (s', post(\mathcal{Z}, e) \uparrow_{\psi})$ if $post(\mathcal{Z}, e) \neq \emptyset$.*

(b) *For a clock zone $\mathcal{Z}$, there exists a delay transition (q, ϵ, q'), where $q = (s, \mathcal{Z}) \xrightarrow{\epsilon}_{ZG} q' = (s, \mathcal{Z}')$ and $\mathcal{Z}' = \mathcal{Z} \uparrow_{R(s)} \cap Inv(s)$ where $\mathcal{Z}'$ is called a time successor of zone $\mathcal{Z}$.*

8.3.5 Deciding Reachability in (Derivative) Multi-timed Zone Graphs

Derivative Multi-timed Zone Graphs (DMZG) can be used as the basis for a reachability checking algorithm (see Section 5.3.4). The algorithm C.1 follows the same structure as the algorithm defined in Section 7.4.3 (see Algorithm 7.1). The Algorithm C.1 formalizes this process to compute the reachability zone graph for a state q_0. The algorithm constructs a finite symbolic zone graph ($\mathsf{DMZG}_{Extra^{+}_{LU}}(\mathscr{A})$), given a DMTA ($\mathscr{A}$). However, since the multi-timed bisimulation algorithm used here uses two DMTA ($\mathscr{A}$ and $\mathscr{B}$), we must first construct a finite symbolic zone graph ($\mathsf{DMZG}_{Extra^{+}_{LU}}(\mathscr{C})$), given the parallel composition of two DMTA ($\mathscr{C}$ is the parallel composition of $\mathscr{A}$ and $\mathscr{B}$) with the same actions $\Sigma_{\mathscr{A}} = \Sigma_{\mathscr{B}}$, but disjoint clocks $X_{\mathscr{A}} \cap X_{\mathscr{B}} = \emptyset$).

proposition 26 (Completeness)**.** *Let $\theta = (s_0, v_0, t_0) \xrightarrow{\vec{d}_0, a_0} (s_1, v_1, t_1) \xrightarrow{\vec{d}_1, a_1} (s_2, v_2, t_2) \ldots \xrightarrow{\vec{d}_{n-1}, a_{n-1}} (s_n, v_n, t_n)$ be an initial (but not necessarily accepting) run of $\mathsf{MLTS}(\mathscr{A}, \tau)$, for some $\tau \in Rates$. Then, for any state (s_i, v_i, t_i), where $0 \leq i \leq n$, appearing in this run, there exists a symbolic zone $(s_i, \mathcal{Z}_i)$ added in Q such that $v_i \in \mathcal{Z}_i$.*

Proof. The proof follows the same lines as the proof in 16. The complete proof for this proposition is given in Appendix C (See Propostion 33).

□

The above proposition tells us that Algorithm C.1 over-approximates reachability. Now, we can establish the termination of Algorithm C.1, because there are finitely many $Extra^{+}_{LU}$ zones [BBLP06].

8.3.6 (Derivative) Refinement Algorithm

In algorithm 7.2, we describe a refinement algorithm (see Section 5.3.10) to compute the multi-timed bisimulation from their zone graph $\mathsf{DMZG}(\mathscr{C})$, where $\mathscr{C} = \mathscr{A} \parallel \mathscr{B}$. The state space Q of $\mathsf{DMZG}(\mathscr{C})$ is initially divided into zones that over-approximate the co-reachable states of $\mathscr{A}$ and $\mathscr{B}$. Thus, we introduce the multi-time and discrete refinement operators.

Definition 106 (**(Derivative) Multi-Time Refinement Operator**)**.** *Let X be a finite set of clocks. Let $(s, \mathcal{Z})$ and $(s, \mathcal{Z}')$ be two zones of the same location, then:*

$$TimePred_{\uparrow_{\psi}}(\mathcal{Z}, \mathcal{Z}') = \{v \in \mathcal{Z} \mid \exists\, \vec{d} \in \mathbb{R}^{Proc}_{>0},\ \exists\, \tau \in Rates,\ \exists\, t, t'' > 0,\ \ t \leq t'',$$

$$\vec{d} = \tau(t'') - \tau(t),\ (v + \vec{d}) \in \mathscr{Z}',\ and\ \forall t', t \leq t' \leq t'', \exists \vec{d}', \vec{d}' = \tau(t') - \tau(t)$$

$$then\ (v + \vec{d}') \in (\mathscr{Z} \cup \mathscr{Z}'),\ and\ \exists \psi \in \Psi(X), (\tau, t') \models \psi\}$$

Then, $TimePred_{\uparrow_\psi}(\mathscr{Z}, \mathscr{Z}')$ is the set of valuations in the clock zone $\mathscr{Z}$ from which a valuation of $\mathscr{Z}'$ can be reached through the elapsing of time, while staying in $(\mathscr{Z} \cup \mathscr{Z}')$ without entering any other clock zones than $\mathscr{Z}$ and $\mathscr{Z}'$ (i.e., $\mathscr{Z} \cup \mathscr{Z}'$). The $TimePred_{\uparrow_\psi}(\mathscr{Z}, \mathscr{Z}')$ operator refines $\mathscr{Z}$ by selecting the states that can reach $\mathscr{Z}'$.

proposition 27. *Let $q = (s, \mathscr{Z})$, $q' = (s, \mathscr{Z}') \in Q$ be two zones, then $TimePred_{\uparrow_\psi}$ $(\mathscr{Z}, \mathscr{Z}')$ is a clock zone.*

Proof. The proof follows the same lines as the proof in 17. The complete proof for this proposition is given in Appendix C (See Propostion 34).

□

For the discrete refinement operator, we use the same definition as in 88. The Algorithm C.2 follows the same structure as the algorithm defined in Section 7.4.3 (see Algorithm 7.2). Algorithm C.2 consists of two steps as in 7.2: The initial phase, is responsible for keeping pairs of states in zones so that each pair of states $((s_{\mathscr{A}}, s_{\mathscr{B}}), (v_{\mathscr{A}}, v_{\mathscr{B}}))$ from the same zone q have the same action. The refinement phase consists of computing the (derivative) timed predecessors (see Definition 107 below) and the discrete action predecessors (see Definition 90) until a set of convex zones is reached.

Definition 107 (Time Refinement). *Let Π be a set of zones and $q = (s, \mathscr{Z})$, $q' = (s', \mathscr{Z}')$ be two zones in Π with $s = s'$. Then for the delay transitions, the refinement function is defined as follows:*

$TimeRefines(\mathscr{Z}, \Pi) = \{TimePred_{\uparrow_\psi}(\mathscr{Z}, \mathscr{Z}') \mid \mathscr{Z}' \in \Pi, q \xrightarrow{\uparrow}_{ZG} q'\}$.

proposition 28. *Let $(s, \mathscr{Z})$ be a zone of Π and let $(e_{\mathscr{A}}, e_{\mathscr{B}})$ be an edge of the $DMZG(\mathscr{C})$, then each of $TimeRefine(\mathscr{Z}, \Pi)$ and $DiscreteSigRefine(\mathscr{Z}, \Pi)$ forms a set of convex zones $\mathscr{Z}$ in Π.*

Proof. The proof follows the same lines as the proof in 18. The complete proof for this proposition is given in Appendix C (See Propostion 35).

□

Given a set of zones Π, Algorithm C.2 computes the states $((s_{\mathscr{A}}, s_{\mathscr{B}}), \mathscr{Z})$ from Π that are bisimilars in particular whether the initial state $((s^0_{\mathscr{A}}, s^0_{\mathscr{B}}), \mathscr{Z}_0)$ are bisimilar.

proposition 29. *Let Π be an initial set of zones and $q = (s, \mathscr{Z})$ be a zone in Π. Let Π' be a final stable set of zones. Let $(s_{\mathscr{A}}, v_{\mathscr{A}})$ and $(s_{\mathscr{B}}, v_{\mathscr{B}})$ be two states in q, then $(s_{\mathscr{A}}, v_{\mathscr{A}}) \approx (s_{\mathscr{B}}, v_{\mathscr{B}})$ iff $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}} \cup v_{\mathscr{B}}) \in q'$, where q' is a zone of the final set of stable zones Π'.*

We reduce the acceptance problem for LB-ATM to the problem of deciding whether two DMTA are multi-timed bisimilar.

Theorem 64. *Deciding multi-timed bisimulation between two DMTA is EXPTIME-complete.*

The proof follows the same lines as the proof in 59.

8.4 A (Derivative) Multi-Timed Modal Logic

Here, we propose DML_ν, an extension of ML_ν over clock derivative. We define the syntax and the semantics of ML_ν over executions of MLTS with such a semantics, and we show that its model checking problem against DML_ν is EXPTIME-complete.

8.4.1 Syntax of DML_ν

We first present the syntax of the logic DML_ν. The logic *DML*$_\nu$ is defined by the following formulas:

Definition 108. *Let* Σ *be a finite alphabet,* X *be a finite set of clocks,* $Proc$ *be a set of processes,* $\pi\colon X \to Proc$ *be a function that maps each clock to a process,* $\dot{X}$ *be a finite set of time clocks derivatives, and Id the set of proposition identifiers. The formulae of DML*$_\nu$ *over* Σ*,* X *and Id are defined by the grammar:*

$$\varphi ::= \ true \mid false \mid \varphi_1 \ \wedge \ \varphi_2 \mid \varphi_1 \ \vee \ \varphi_2 \mid [a]\varphi \mid \langle a\rangle\varphi \mid$$

$$\exists\varphi \mid \forall\varphi \mid x^p \ \underline{in} \ \varphi \mid \phi \mid \ x + c \sim y + d \mid \psi \mid Z$$

where $a \in \Sigma$*,* $p \in Proc$*,* $p = \{x, y\}$*,* $x, y \in X$*,* $c, d \in \{0, \cdots, k\}$*,* k *a non-negative integer,* $\sim \in \{=, >, \geq, <, \leq\}$*,* $Z \in Id$*,* $\phi \in \Phi(X)$ *a clock constraint,* $\psi \in \Psi(X)$ *a rate constraint,* $[a]\varphi$*,* $\langle a\rangle\varphi$ *are two modalities of the logic, and* $\exists\varphi$ *and* $\forall\varphi$ *are the two timed modalities. Note that we can only compare two clocks of the same process, but we can compare two clock derivatives of different processes.*

The identifiers Id are specified by a declaration environment, $\mathscr{D}$ assigning a DML_ν formula to every identifier in order to define properties with maximal fixpoints. A declaration is noted by $Z \overset{\text{def}}{=} \varphi$ for $\mathscr{D}(Z) = \varphi$.

8.4.2 Semantics of DML_ν

Let $\mathscr{A}$ be a DMTA over *Proc* and $\tau \in$ Rates and assume that $\mathsf{MLTS}(\mathscr{A}, \tau) = (Q, q_0, \Sigma, \to_{mlts})$ gives its semantics. Now, we interpret DML_ν formulas over extended states. An extended state over Q is a pair (q, μ), where $q \in Q$ is a MLTS state (Definition 5) and μ a valuation for the formula clocks in X. An extended state satisfies an identifier Z if it belongs to the maximal fixpoint of the equation $Z = \mathscr{D}(Z)$. The formal semantics of DML_ν formulas interpreted over $\mathsf{MLTS}(\mathscr{A}, \tau)$ is given by the satisfaction relation, $\models$ defined as the largest relation satisfying the equivalences in 108.

Definition 109. *Let Σ be a finite alphabet, X be a finite set of clocks and $Proc$ be a set of processes. The semantics of formulae in ML_ν is implicitly given with respect to a given MLTS inductively as follows:*

$$
\begin{array}{lcl}
(q,\mu) \models & true & \Leftrightarrow true \\
(q,\mu) \models & false & \Leftrightarrow false \\
(q,\mu) \models & \varphi_1 \wedge \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \textit{ and } (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \varphi_1 \vee \varphi_2 & \Leftrightarrow (q,\mu) \models \varphi_1 \textit{ or } (q,\mu) \models \varphi_2 \\
(q,\mu) \models & \phi & \Leftrightarrow \mu \models \phi \textit{ for } \phi \in \Phi(X) \\
(q,\mu) \models & \psi & \Leftrightarrow \mu \models \phi \textit{ for } \psi \in \Psi(X) \\
(q,\mu) \models & [a]\varphi & \Leftrightarrow \forall q \xrightarrow{a}_{mlts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & \langle a \rangle \varphi & \Leftrightarrow \exists q \xrightarrow{a}_{mlts} q', (q',\mu) \models \varphi \\
(q,\mu) \models & x\ \underline{in}\ \varphi & \Leftrightarrow (q,\mu[x \to 0]) \models \varphi \\
(q,\mu) \models & \exists \varphi & \Leftrightarrow \exists \vec{d} \in \mathbb{R}^{Proc}_{\geq 0}, \exists q' \in Q, \textit{ such that } q \xrightarrow{\vec{d}}_{mlts} q', \\
 & & \quad (q,\mu +_\pi \vec{d}) \models \varphi \\
(q,\mu) \models & \forall \varphi & \Leftrightarrow \forall \vec{d} \in \mathbb{R}^{Proc}_{\geq 0}, \forall q' \in Q, \textit{such that } q \xrightarrow{\vec{d}}_{mlts} q', \\
 & & \quad (q,\mu +_\pi \vec{d}) \models \varphi \\
(q,\mu) \models & x + c \sim y + d & \Leftrightarrow \mu(x) + c \sim \mu(y) + d \\
(q,\mu) \models & Z & \textit{the maximal fixpoint in } \mathscr{D}(Z)
\end{array}
$$

Two formulae are equivalent iff they are satisfied by the same set of extended states in every MLTS.

Definition 110. *Let $\mathscr{A}$ be a DMTA and $\varphi \in DML_\nu$, then $\mathscr{A} \models \varphi$ iff $\forall \tau \in Rates$, MLTS$(\mathscr{A},\tau) \models \varphi$.*

Theorem 65. *Let $Proc$ be a set of processes. Let $\mathscr{M} = (Q, q_0, \Sigma, \to_{mlts})$ be a MLTS and q_1, q_2 be multi-timed bisimilar states in Q. Let μ be a clock valuation for the formula clocks in X, then the extended states (q_1,μ) and (q_2,μ) satisfy exactly the same formulae in ML_ν.*

The proof follows the same lines as the proof in 60.

Theorem 66. *Let $\mathscr{A}_1$ and $\mathscr{A}_2$ be two DMTA and $\forall$ φ formula in the logic DML_ν. If ($\mathscr{A}_1 \models \varphi$ iff $\mathscr{A}_2 \models \varphi$) then $\mathscr{A}_1 \approx \mathscr{A}_2$.*

Proof. (sketch). A proof of this theorem may be obtained from the characteristic property of TA [LLW95]. □

proposition 30. *Given two DMTA $\mathscr{A}_1$ and $\mathscr{A}_2$, if $\mathscr{A}_1 \approx \mathscr{A}_2$ and $\mathscr{A}_1 \models \varphi$ then $\mathscr{A}_2 \models \varphi$.*

8.4.3 Examples of Properties

Here, we use DML_ν formulas to express multi-timed properties.

Example 41. *Consider the DMTA $\mathscr{M}$ described in Figure 8.1. Let $Proc = \{p, q\}$ be a set of processes, where $p = \{x\}$, $q = \{y\}$, $x, y \in X$ and $\dot{x}$, $\dot{y} \in \dot{X}$. The initial state (q_0, μ_0) (i.e., $q_0 = (S_0, v_0)$) satisfies the following DML_ν formula φ:*

$$\varphi \;=\; y\ \underline{in}\ \exists((\,2 \geq y \geq 0 \;\wedge\; \dot{x} > \dot{y} \;\wedge \langle a\rangle\, S_1) \wedge\; (x \geq 1 \;\wedge\; \dot{y} = 1) \;\wedge\; \langle a\rangle\, S_1)$$

Intuitively, this formula means that the action a can be performed by the process q after a delay between 0 and 2, and satisfying the rate constraint $\dot{x} > \dot{y}$, for instance 1 and the action a can be performed by the process p after a delay 1 time units, and satisfying the rate constraint $\dot{y} > 1$.

Example 42. *Consider the DMTA $\mathscr{A}$ described in Figure 8.8. The state (q_1, μ_1) (i.e., $q_1 = (S_1, v_1)$) satisfies the following DML$_v$ formula φ:*

$$Z_{S_1} \;=\; x^p\ \underline{in}\ \exists(x^p \leq 10 \;\wedge \dot{x} > 1 \;\wedge \dot{y} = 1 \;\wedge\; \langle b\rangle\, Z_{S_1}) \wedge [b]\ (x^p \leq 10 \;\wedge \dot{x} > 1 \;\wedge \dot{y} = 1 \;\wedge\; x^p\ \underline{in}\ Z_{S_1}) \wedge \forall\, Z_{S_1} \;\vee\; y^q\ \underline{in}\ \exists(y^q \leq 9 \;\wedge\; \langle b\rangle\, S_0) \wedge x^p\ \underline{in}\ (\langle b\rangle\, S_0)$$

Intuitively, this formula means that the action b can be performed by the process p before a delay 10 time units (self-loop) or the action b can be performed by the process q before a delay 9 time units.

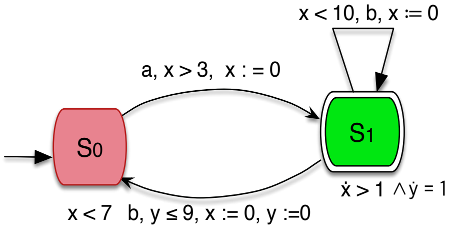


Figure 8.8: A Multi-timed Automata $\mathscr{A}$

Theorem 67. *The model checking problem of DML$_v$ on DMTA is EXPTIME-complete.*

The proof follows the same lines as the proof in 62.

8.4.4 Satisfiability Checking

The satisfiability checking problem, which is the dual of the model checking problem, is to check whether a given φ formulae is satisfied by a multi-timed automaton $\mathscr{A}$. Formally, let φ be a DML$_v$ formula and $\mathscr{A}$ be a DMTA, then the satisfiability problem ($\mathscr{A} \models \varphi$) is decidable.

8.5 Application of DMTA and DMLv

In this section, we present the experimental results related to our (derivative) alternative semantics. We introduce the FireWire - IEEE 1394 protocol and show how this protocol can be modeled as a DMTA. We also describe the properties of this protocol in DML$_v$.

8.5.1 FireWire - IEEE 1394 Protocol

FireWire - IEEE 1394 is a high-performance serial communications bus that supports peer-to-peer, synchronous, and asynchronous data transfers between multimedia devices. The bus is hot-plug-and-play and can work with a wide variety of devices. The architecture protocol is organized in arbitrary topologies where nodes are connected by cables (i.e., channels). The physical topology of the bus is represented by a logical tree structure, where the root (leader) mediates access to the bus. The bus configuration consists of three phases: (1) bus initialization, (2) tree identification, and (3) self-identification. In the second phase, the tree identification protocol is initiated to select one of the nodes as the root of the tree. If the tree topology changes (i.e., a device joins/leaves the tree), a reset occurs and a new leader election process is restarted. The tree discovery protocol is initiated by the leaf nodes of the tree. Leaf nodes start by sending $parent\_signal$ requests to their nearest neighbor nodes. When a node receives a $parent\_signal$ request, it is marked as a child and sends a $child\_signal$ request. At the end of the second phase, two neighboring nodes may be sending $parent\_signal$ to each other. This conflict is resolved by forcing both nodes to resend requests after a random delay. Figure 8.9 gives an overview of the FireWire protocol, inspired by the case study [DLL$^+$11], where the considered topology is represented by devices (nodes) and cables (channels) and their interactions. The nodes and channels (one wire for each direction) are modeled by TA.

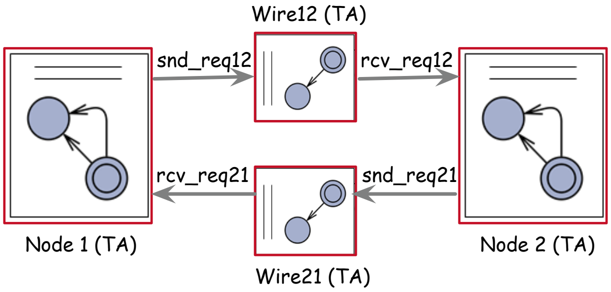


Figure 8.9: FireWire component composition

Here, we consider the FireWire model described in [DLL$^+$11] and [CAS01]. We show how DMTA can be used to model FireWire with independent clocks and rate constraints.

8.5.2 Modeling the FireWire - IEEE 1394 Protocol in DMTA

In [DLL$^+$11] and [CAS01], TA were used to model the FireWire protocol. These TA consists of an automaton for each node, an automaton for each wire (cable), and a parallel composition between them. The node automaton determines whether a node is a root or a child. The wire automaton models the unidirectional communication between nodes. Here, we extend the FireWire protocol with independent clocks. Nodes can be modeled as MTA. For each node, we can construct a DMTA that is able to process the requests of the leader election within the appropriate time. Here we model with our DMTA a simplification of node automata taken from [CAS01], but without the wire automata. For the Node1 (or Node2) of Figure 8.9, this is as follows

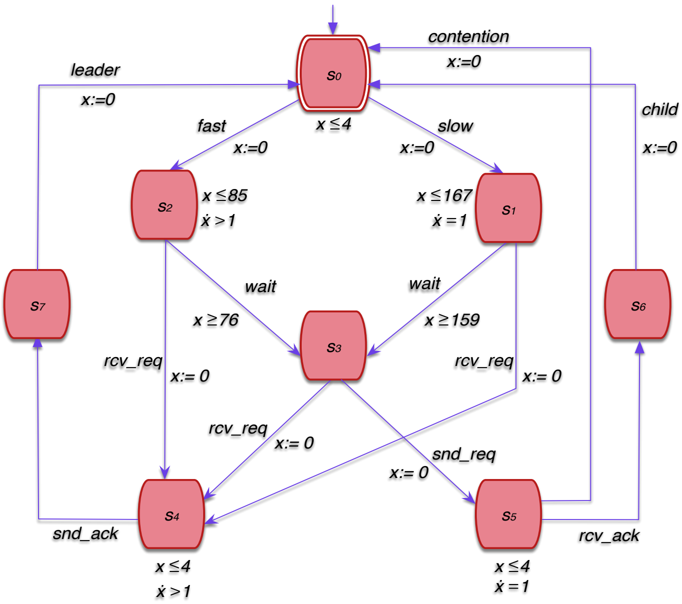


Figure 8.10: A simplified modeling of a node (FireWire Protocol from [DCBB19]) in DMTA

Figure 8.10 below shows a DMTA $\mathscr{A}$ for the $Node1$ of Figure 8.9 with the finite alphabet Σ ={ rcv_req, snd_req, rcv_ack, snd_ack, $wait$, $fast$, $slow$, $leader$, $child$, $contention$ }, $Proc = \{p\}$, independent clocks $X = \{x\}$. The actions $fast$ or $slow$ correspond to the start of the node (node1). Then, in location S_0 there is a choice between moving to location S_2 with action fast (faster clock), invariant $x \leq 85$ and rate constraint $\dot{x} > 1$ or location S_1 with action slow (slower clock), invariant $x \leq 167$ and rate constraint $\dot{x} = 1$. In both cases, we move to a location where invariant and rate constraints are satisfied. the processors. In location S_3, we also have a choice between moving to location S_4 with action rcv_req, invariant $x \leq 4$ and rate constraint $\dot{x} > 1$ or location S_5 with action snd_req, invariant $x \leq 4$ and rate constraint $\dot{x} = 1$.

8.5.3 Properties of the FireWire Protocol in DML$_\nu$

Example 43. *Consider the DMTA $\mathscr{A}$ described in Figure 8.10. The initial state (q_0, μ_0) (i.e., $q_0 = (S_0, \nu_0)$) satisfies the following DML$_\nu$ formula (property) φ:*

$$\varphi \ = \ x \ \underline{in}\ \exists((x \leq 85 \ \wedge \ \dot{x} > 1 \ \wedge \ \langle fast \rangle \ S_2) \vee \exists(x \leq 167 \ \wedge \ \dot{x} = 1 \ \wedge \ \langle slow \rangle \ S_1))$$

Intuitively, this formula means that the action $fast$ can be performed by the node 1 (p) before a delay 4 *time units and thereby to reach the* S_2 *location where the invariant* $x \le 85$ *and rate constraint* $\dot{x} > 1$ *are satisfied, or the action* $slow$ *can be performed before a delay* 4 *time units and thereby to reach the* S_1 *location where the invariant* $x \le 167$ *and rate constraint* $\dot{x} = 1$ *are satisfied,*

8.5.4 Fischer's Protocol

The Fischer protocol system [ACH$^+$95] consists of several processes, all of which attempt to access a shared resource. Access to this resource is controlled by Fischer's mutual exclusion algorithm [ACH$^+$95]. In TA, each process is modeled by four locations [DOTY96]. An additional automaton models access to the critical section by label synchronization with the automatons [DOTY96]. Here we consider Fischer's protocol described in [ACH$^+$95]. We show how DMTA can be used to model Fisher's protocol with independent clocks and rate constraints.

8.5.5 Modeling the Fischer's Mutual Exclusion Protocol in DMTA

In [DOTY96] and [RMS16], TA was used to model Fischer's protocol. These TA consists of an automaton for each process and an automaton to model access to the critical section. Here, we extend Fisher's protocol with independent clocks for each process and rate constraints. Figure 8.11 shows a protocol process:

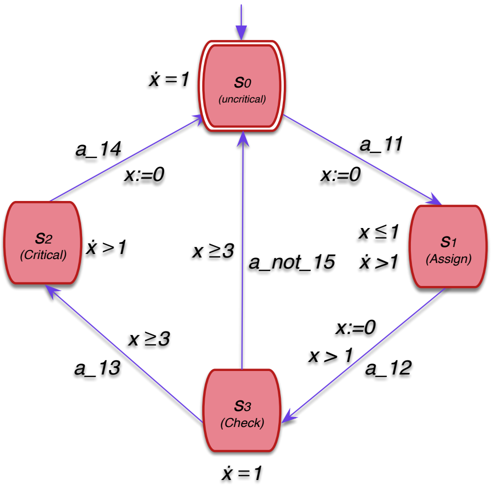


Figure 8.11: A process p1 (Fischer's protocol from [DOTY96]) in DMTA

Figure 8.11 below shows a DMTA $\mathscr{A}$ for the $process1$ of the Fischer's Protocol with the finite alphabet Σ ={ a_11, a_12, a_13, a_14, a_not_15}, $Proc = \{p1\}$, independent clocks $X = \{x\}$. Then, in location S_0 with rate constraint $\dot{x} = 1$ and action a_11, there exists only one transition to location S_2. In location S_1 with invariant $x \ge 1$, rate constraint $\dot{x} > 1$ and action a_12, there exists a transition to location S_3. In location S_3 with rate constraint $\dot{x} = 1$, we have a choice between moving to location S_2 with action a_{13}, rate constraint $\dot{x} > 1$ or location S_0 with action a_not_15 and rate constraint $\dot{x} = 1$. In both cases, we move to a location where invariant and rate constraints are satisfied. Finally, in location S_2, there exists only one transition to location S_0 with action a_{14} and rate constraint $\dot{x} = 1$.

8.5.6 Properties of the FireWire Protocol in DML_ν

Example 44. *Consider the DMTA $\mathscr{A}$ described in Figure 8.11. The initial state (q_0, μ_0) (i.e., $q_0 = (S_0, \nu_0)$) satisfies the following ML_ν formula (property) φ:*

$$\varphi \;=\; x\ \underline{in}\ \exists(x \leq 1 \;\wedge\; \dot{x} > 1 \;\wedge\; \langle a_{11}\rangle\ S_1)$$

Intuitively, this formula means that the action a_{11} can be performed by the process $p1$ before a delay 4 time units and thereby to reach the S_1 location where the invariant $x \leq 1$ and rate constraint $\dot{x} > 1$ are satisfied,

8.6 Implementation

In this section, we present the functionalities and details related to the implementation of our algorithms of model checker and multi-timed bisimulation. We also present some experimental results.

8.6.1 Implementation of the Multi-timed bisimulation Algorithms

We have implemented a tool (called MUTES[1]) based on the algorithms presented in section 8.3.5 (parallel composition of two DMTA ($\mathscr{A}$ and $\mathscr{B}$), symbolic (derivative) multi-zone graph, and multi-timed bisimulation). We have used the *Java* 8 version in the implementation of our tool. As input, our tool receives two UPPAAL file formats (UPPAAL model) [UPP]. UPPAAL is a tool for modeling, simulation, and verification of networks of TA extended with data types, user functions, clocks, and synchronous communication channels [UPP]. A UPPAAL model consists of several components: (1) Declarations: consisting of global declarations, (2) Templates: describing processes as timed automata, (3) System Declarations: consisting of process instantiations. In addition, UPPAAL supports three types of formats: XML, XTA, and TA. However, to be able to represent our DMTA, we need to add to the (original) UPPAAL model (i.e., inside the Templates component) the ability to define processes, rate constraints, independent clocks, and map clocks to processes. Either by using the XML file (see Figure 8.12) or by using the XTA language (see figure 8.13). The (DMTA) UPPAAL model of figures 8.12 and 8.13 is shown in Figure 8.14 (graphical user interface).

We have used the ANTLR parser generator [Par13] to read the XML files generated by UPPAAL, which are used by our tool to identify the independent clocks, transitions, invariants, rate constraints, guards, and build automata according to

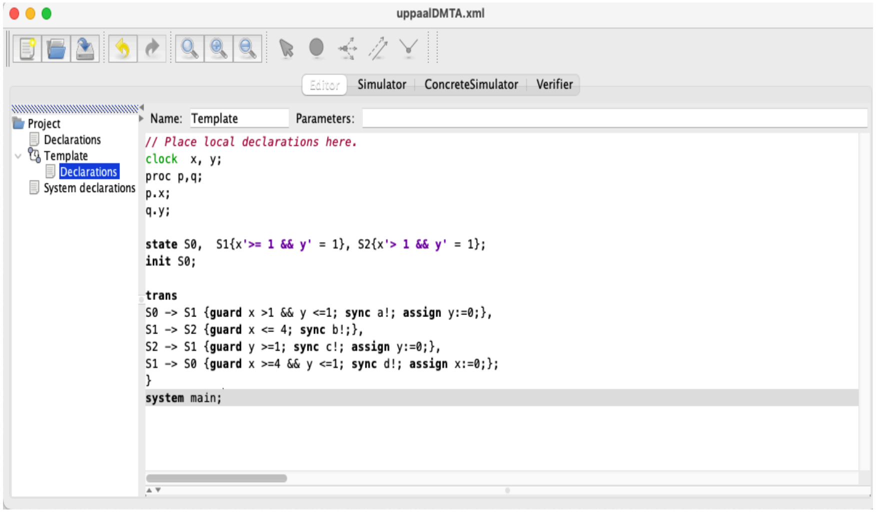
[1] MUTES source code is available: https://github.com/jortizve/MUTES

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN' 'http://www.it.uu.se
    /research/group/darts/uppaal/flat-1_2.dtd'>
<nta>
  <declaration>// Place global declarations here.</declaration>
  <template>
    <name x="5" y="5">Template</name>
    <declaration>// Place local declarations here.
clock  x, y;
proc p,q;
p.x;
q.y;
</declaration>
    <location id="id0" x="-450" y="34">
      <name x="-460" y="0">S0</name>
    </location>
    <location id="id1" x="-229" y="34">
      <name x="-239" y="0">S1</name>
      <label kind="invariant" x="-280" y="59">x' &gt;= y' &amp;&amp; y' = 1</label>
    </location>
    <location id="id2" x="-25" y="34">
      <name x="-35" y="0">S2</name>
      <label kind="invariant" x="-59" y="51">x' &gt; y' &amp;&amp; y' = 1</label>
    </location>
    <init ref="id0"/>
    <transition>
      <source ref="id0"/>
      <target ref="id1"/>
      <label kind="guard" x="-382" y="-59">x&gt; 1 &amp;&amp; y&lt;=1</label>
      <label kind="synchronisation" x="-356" y="-42">a!</label>
      <label kind="assignment" x="-331" y="-42">y:=0</label>
      <nail x="-339" y="-25"/>
      <nail x="-331" y="-25"/>
    </transition>
    ...
  </template>
  <system>// Place template instantiations here.
Process = Template();
// List one or more processes to be composed into a system.
  ...
</nta>
```

Figure 8.12: XML file

the syntax of DMTA. After parsing our two xml input files (i.e., two MTA models), our tool: (1) generates a DMTA C, which is a parallel composition of two DMTA input files (A and B) with the same actions $\Sigma_{\mathscr{A}} = \Sigma_{\mathscr{B}}$, but disjoint clocks $X_{\mathscr{A}} \cap X_{\mathscr{B}} = \emptyset$, (2) build a finite symbolic multi-timed zone graph given the DMTA $\mathscr{C}$ (i. e., MZG$_{Extra^{+}_{LU}}(\mathscr{C})$), (3) and perform a refinement algorithm to check the (derivative) multi-timed bisimulation between the two DMTA models. Figure 8.15 shows an overview of our tool (MUTES).

In the building of the multi-timed zone graph described above (2), we used difference bound matrices (DBM) [Dil90] [BY04] and difference bound matrices with Clock derivatives (DBMC) as a data structure to represent and manipulate (Derivatives) multi-timed zones. The UPPAAL grammar has been extended to use the semantics of DMTAs such as rates constraints and clock derivatives (see Figure 8.16).



```
// Place local declarations here.
clock  x, y;
proc p,q;
p.x;
q.y;

state S0,  S1{x'>= 1 && y' = 1}, S2{x'> 1 && y' = 1};
init S0;

trans
S0 -> S1 {guard x >1 && y <=1; sync a!; assign y:=0;},
S1 -> S2 {guard x <= 4; sync b!;},
S2 -> S1 {guard y >=1; sync c!; assign y:=0;},
S1 -> S0 {guard x >=4 && y <=1; sync d!; assign x:=0;};
}
system main;
```

Figure 8.13: XTA format

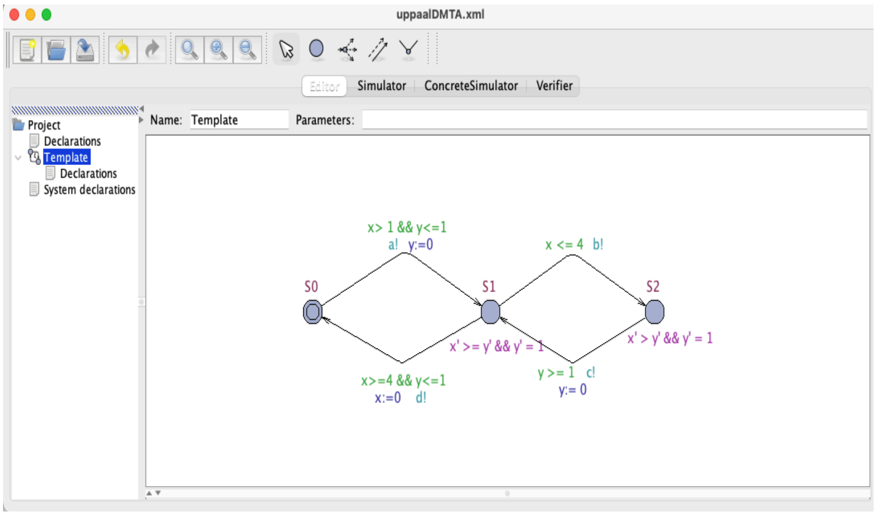


Figure 8.14: UPPAAL GUI

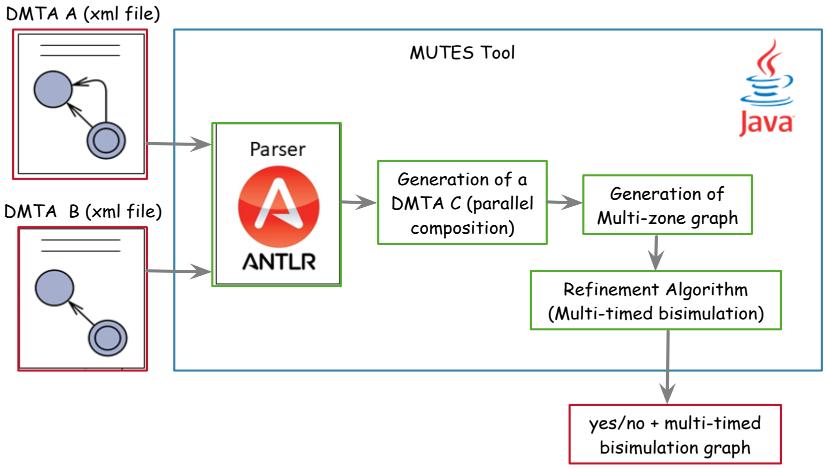

DMTA A (xml file)
DMTA B (xml file)
MUTES Tool
Java
Parser
ANTLR
Generation of a DMTA C (parallel composition)
Generation of Multi-zone graph
Refinement Algorithm (Multi-timed bisimulation)
yes/no + multi-timed bisimulation graph

Figure 8.15: MUTES Tool

8.6.2 Experimental Evaluation

In this section, we present experimental results to illustrate the practical performance of our tool with various case studies. Due to the lack of real case studies related to independent clocks and rate constraints, we have used some real case studies taken from the community benchmarks. Our studies come from UPPAAL specifications of these cases are available at https://github.com/farkasrebus/XtaBenchmarkSuite (community benchmarks). For each case study, we consider the largest and most important automaton (or process in UPPAAL) from the automata network, and we define a single process where all clocks belong to this process. We also set the derivative of the clocks to 1.

Gear Control (GC). The GC models a simple gear controller for vehicles [LPY98]. The GC model contains 24 states, of which 10 have invariants. All invariants are of the form $x \leq c$ for a clock x and constant c. There are 30 transitions, of which 2 transitions have guards of the form $x < c$ and 2 transitions have guards of the form $x \geq c$, for some clock x and constant c.

Collision Avoidance (CA). The CA case models a protocol where different agents want to get access to Ethernet through a shared channel [JLS96]. The CA model has 6 states and 12 transitions, of which 9 transitions have guards of the form $x == c$ and 4 transitions have guards of the form $x < c$, for some clock x and constant c.

Train Gate Controller (TGC). The TGC models a railway system that controls access to a bridge for several trains [UPP]. The bridge is a shared resource that may be accessed only by one train at a time. The TGC model has 14 states and all of them have invariants. All invariants are of the form $x < c$ for a clock x and constant c. There are 18 transitions, of which 4 transitions have guards of the form $x < c$ and 4 transitions have guards of the form $x > c$, for some clock x and constant c.

A combined Gear control (CGC). The CGC models a (manually) combined gear controller for vehicles [LPY98]. The CGC model contains 85 states, of which 20 have

```
NTA       : ’nta’ ;
DECLARATION   : ’declaration’ ;
TEMPLATE    : ’template’ ;
LOCATION    : ’location’ ;
NAME      : ’name’ ;
TRANSITION    : ’transition’ ;
SOURCE      : ’source’ ;
TARGET      : ’target’ ;
NAIL      : ’nail’ ;
PROCESS         :   ’proc’;
...
RATE             :  ’"rate"’;
INVARIANT   : ’"invariant"’ ;
SYNCHRONIZE   : ’"synchronisation"’ ;
GUARD     : ’"guard"’ ;
ASSIGNMENT    : ’"assignment"’ ;
SELECT      : ’"select"’ ;
...
// Atomic types
INT       : ’int’ ;
CLOCK     : ’clock’ | ’ clock’ ’;
CHANNEL     : ’chan’ ;
BOOL      : ’bool’ ;
DOUBLE      : ’double’ ;
SCALAR      : ’scalar’ ;
...
// Operations
ASSIGN      : ’=’
        | ’:=’
        | ’+=’
        |   ’-=’
        |   ’*=’
        |   ’/=’
        |   ’%=’
              | ’|=’ | ’&=’ | ’^=’ | ’<<=’ | ’>>=’
              ;
```

Figure 8.16: Part of MUTES grammar

invariants. All invariants are of the form $x \leq c$ for a clock x and constant c. There are 120 transitions, of which 10 transitions have guards of the form $x < c$, and 10 transitions have guards of the form $x \geq c$, for some clock x and constant c.

Using these original models, we consider an experimental framework for running our tool. We use an existing technique called mutation testing (MT) [JH11], which creates mutants of the system by injecting artificial defects via predefined mutation operators. Then one can run tests on the system and a mutant and compare the obtained results: if they are different, the mutant is said to be *killed* (or *distinguished*) by these tests. It is possible to measure the effectiveness of the tests by the *mutation score*, i.e. the percentage of mutants killed over the total number of mutants generated. MT has long been a mostly code-based approach [PM10, FA14, Off11], but model-based mutation testing (MBMT) helps to automatically identify defects related to missing functionality and misinterpreted specifications [BG85] that are difficult to identify through code-based testing [How76, VM97]. Not all mutants are useful, however. Some of them may be equivalent, i.e. they exhibit the same behavior as the original system despite their syntactic difference [PKZ+18]. Therefore, no test case can distinguish the mutant from the original system. Similarly, duplicate mu-

	GC	CA	TGC	CGC
TMI	13	9	14	36
TAD	501	26	179	1,625
SMI	12	2	12	27
CXL	0	1	4	4
CXS	2	1	4	6
CCN	2	2	8	10
Total	533	41	222	1713

Table 8.1: Number of generated mutants per operator

tants are mutants that exhibit the same behavior as other mutants [PJHL15, PKZ$^+$18]. Preventing and removing such useless mutants reduces the computational cost of MT and builds more confidence in mutation scores. Recently, Basile et. al. [BtBCL20] tackled the equivalent mutant problem for TA. They defined mutation operators that prevent mutants from refining the original system [BtBCL20, BBL$^+$22]. Basile *et al.* [BtBCL20] proposes six mutation operators on TA (UPPAAL), designed to avoid the generation of mutants that are subsumed by construction. Three of the six mutation operators in [BtBCL20] are time-independent: *transition missing (TMI)* removes a transition, *transition add (TAD)* adds a transition between two states, *state missing (SMI)* removes a state (other than the initial state) and all its incoming/outgoing transitions. The other three time-related operators are *constant exchange Larger (CXL)* increases the constant of a clock constraint, *constant exchange Smaller (CXS)* decreases the constant of a clock constraint, and *clock constraint negation (CCN)* negates a clock constraint. The main idea in [BtBCL20] is to perform a refinement check between the mutant and the system model, using ECDAR (UPPAAL) [LLNN17]. However, this technique does not address duplicate mutants; in our experiments, we used our MUTES tool to detect duplicate mutants and assess the behavioral equivalence between two mutants. Although the duplicate mutant problem is not related to our thesis, we used this problem to measure the scalability and performance of our multi-timed bisimulation algorithm in detecting duplicate mutants. Based on the guidelines of [BtBCL20, BBL$^+$22], we designed and implemented the six operators (see above) that do not generate equivalent mutants but can generate duplicate mutants. The six operators were written in Java 8 and use the ANTLR library to parse the model and generate syntactically correct and non-equivalent mutants (thanks to the operators). The implementation of the six operators, case studies, and results are available on our companion website https://github.com/jortizve/OperatorsImplementation.

We have applied the six operators to the four case studies presented above. This results in a total of 2509 mutants as presented in Table 8.1. We then apply our multi-timed bisimulation algorithm. We ran our experiments on a UBUNTU 21.10 × 86_64 GNU/Linux machine with 16 cores, 2.2 GHz, 32GB RAM.

Table 8.2 reveals that mutant duplicates represent up to 32% of the total number of mutants, justifying the need for duplicate prevention and removal techniques. Our multi-timed bisimulation algorithm has good scalability overall (see Table 8.3).

Case	GC	CA	TGC	CGC
ratio Bisimulation	41/533	12/41	71/222	373/1,713

Table 8.2: Proportion of mutant duplicates

Case	GC	CA	TGC	CGC
average execution time(s)	3.9 (st=0.54)	2.3 (st=0.71)	2.8 (st=1.05)	17.5 (st=3.9)

Table 8.3: The average execution time(s) using multi-timed bisimulation (BI) and the standard deviation (st).

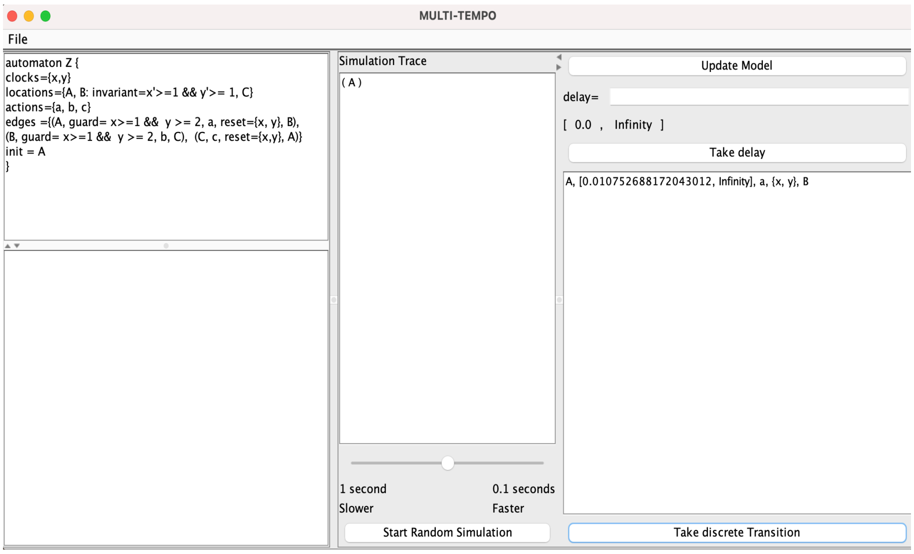
8.6.3 MULTI-TEMPO Tool

We have implemented a tool (called MULTI-TEMPO) based on the semantics presented in section 98 (Semantics of DMTA). We used the *Java* 8 version in the implementation of our tool. MULTI-TEMPO[2] is an integrated tool environment for modeling and simulation of DRTS. MULTI-TEMPO provides a graphical user interface (see Figure 8.17) with two main parts: a description language and a simulator. The description language is a textual command language used to model the behavior of DRTS with clocks, locations, edges, invariants, and rate constraints (see Figure 8.17 (top left)). The simulator allows dynamic executions of a modeled system (see Figure 8.17 (bottom center)). When running the tool, the first step is to describe the distributed and real-time system (see Figure 8.17 (top left)) and then update the model using the Update Model below (see Figure 8.17 (top right)). The second step is to simulate the modeled system using the Start Random Simulation (see Figure 8.17 (bottom center)) or to step through each transition present in the modeled system using the Take Discrete Transition.

A simple example of a modeled system (program) is shown in Figure 8.18. There are five main sections in the specification of an automaton in MULTI-TEMPO: clocks, locations, actions, edges, init.

The clocks lists the possible independent clocks of the modeled system. In Figure 8.18 the clocks are x and y. The locations in our tool are represented by letters. Locations are connected by edges. Locations are labeled with invariants and rate constraints. Invariants are expressions and therefore follow the syntax of expressions. An invariant can be a conjunction of simple conditions on clocks, the constraint must be given by an integer expression. In addition, rate constraints are supported and are declared with invariants. Rate constraints expressions are specified and are part of the conjunction in the invariant. Actions list the possible actions of the modeled system. Locations are connected by edges. Edges are annotated with locations, guards, actions, and reset. Each modeled system must have exactly one initial location. The initial location is described by the expression init. The grammar for modeling DRTS in MULTI-TEMPO is given by (see Figure 8.19):

[2] MULTI-TEMPO source code and jar are available: https://github.com/jortizve/MULTI-TEMPO



```
automaton Z {
clocks={x,y}
locations={A, B: invariant=x'>=1 && y'>= 1, C}
actions={a, b, c}
edges ={(A, guard= x>=1 &&  y >= 2, a, reset={x, y}, B),
(B, guard= x>=1 &&  y >= 2, b, C),  (C, c, reset={x,y}, A)}
init = A
}
```

```
( A )
```

```
A, [0.010752688172043012, Infinity], a, {x, y}, B
```

Figure 8.17: MULTI-TEMPO Tool

```
automaton Z {
clocks={x,y}
locations={A, B: rates=x'>=1 && y'>= 1, C}
actions={a, b, c}
edges ={(A, guard= x>=1 &&  y >= 2, a, reset={x, y}, B),
(B, guard= x>=1 &&  y >= 2, b, C),  (C, c, reset={x,y}, A)}
init = A
}
```

Figure 8.18: A simple program in MULTI-TEMPO

8.6.4 MIMETIC Tool

We have implemented a tool (called MIMETIC[3]) for the verification (automatic model-checking) of DRTS, based on the semantics presented in section 98 (semantics of DMTA) and section 108. We have used *Java* 8 version in the implementation of our tool. MIMETIC provides a graphical user interface (see Figures 8.20 and 8.21) with two main parts: select the XML file (UPPAAL file) and writing a DMLv formula (text box).

We have used the ANTLR parser generator [Par13] to read the XML file generated by UPPAAL and the DMLv formula text box) used by our tool to identify the independent clocks, transitions, invariants, rate constraints, guards, and build automata according to the syntax of DMTA and DMLv. After parsing the xml input files and the DMLv formula, our tool performs a model checking algorithm to verify that a DMTA model satisfies a DMLv formula. Figure 8.22 shows an overview of our tool (MIMETIC).

[3] MIMETIC source code and jar are available: https://github.com/jortizve/MIMETIC

```
model       :   block automaton+;
block       :   statement*;
statement   :   varDeclaration          # VarDeclarationSt
            |   expr                    # ExprSt
            |   ’print’ expr            # PrintSt
            |   ’return’ expr           # ReturnSt
            ;
...
type        :   ’num’ ;
varId       :   IDENTIFIER (’=’ initialiser)? ;
initialiser :   expr ;
automaton   :   ’automaton’ IDENTIFIER
                        ’{’ (locationType | clockType | actionType | edgesType)*
                            initLocation
                            block
                            ’}’ ;
locationType:   ’locations’ ’=’ ’{’ location (’,’ location)* ’}’ ;
clockType   :   ’clocks’ ’=’ ’{’ IDENTIFIER (’,’ IDENTIFIER)* ’}’ ;
actionType  :   ’actions’ ’=’ ’{’ IDENTIFIER (’,’ IDENTIFIER)* ’}’ ;
edgesType   :   ’edges’ ’=’ ’{’ edge (’,’ edge)* ’}’ ;
location    :   IDENTIFIER (’:’ ’invariant’ ’=’ guard)?
                           (’:’ ’rates’ ’=’ ’{’ IDENTIFIER ’=’ expr (’,’ IDENTIFIER
                                 ’=’ expr)* ’}’)?;
initLocation:   ’init’ ’=’ IDENTIFIER ;
edge        :   ’(’ (’source’ ’=’)? IDENTIFIER ’,’
                    (’guard’ ’=’ guard ’,’)?
                    (’action’ ’=’)? IDENTIFIER ’,’
                    (’reset’ ’=’ ’{’ IDENTIFIER (’,’ IDENTIFIER)* ’}’ ’,’ )?
                    (’target’ ’=’)? IDENTIFIER’)’ ;
guard       :   consGuard ((’and’ | ’&&’) consGuard)*
            ;
consGuard   :   expr ;
...
arguments   :   (expr  (’,’ expr)*)? ;

expr        :   op=(’+’ | ’-’) expr         # Unary
            |   expr ’*’ expr               # Mul
            |   expr op=(’+’|’-’) expr      # AddSub
            |   expr op=(’<=’|’>=’) expr    # CompareExpr
            |   IDENTIFIER  ’\’’            # RateExpr
            |   DOUBLE                      # DoubleExpr
            |   IDENTIFIER                  # IdExpr
            |   ’(’ expr ’)’                # ParensExpr
            |   IDENTIFIER ’=’ expr         # AssignExpr
```

Figure 8.19: Part of MULTI-TEMPO grammar

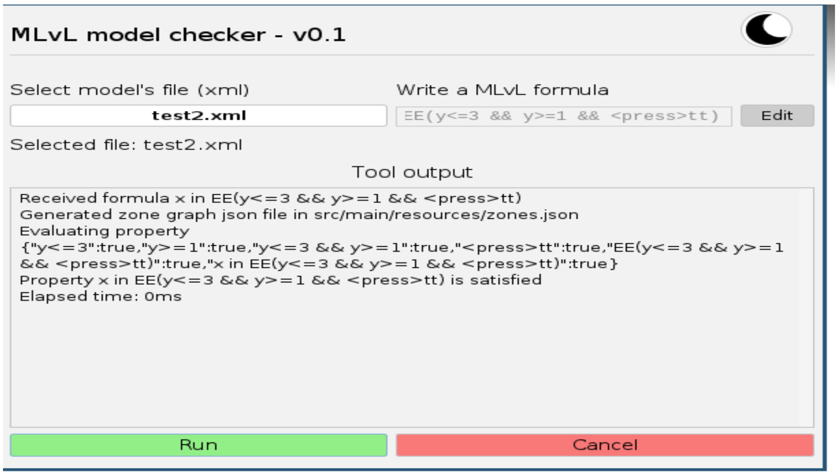
The grammar for modeling and specifying properties in MIMETIC is given by (see Figures 8.23 and 8.24) :

8.7 Strengths and Weaknesses of the Formalisms

This section aims at showing the strengths and weaknesses of the two formalisms presented in this chapter.

8.7.1 Strengths

The main strengths are the incorporation of rate constraints over clock derivatives, an extension of MTA with the notion of rate constraints, a forward reachability



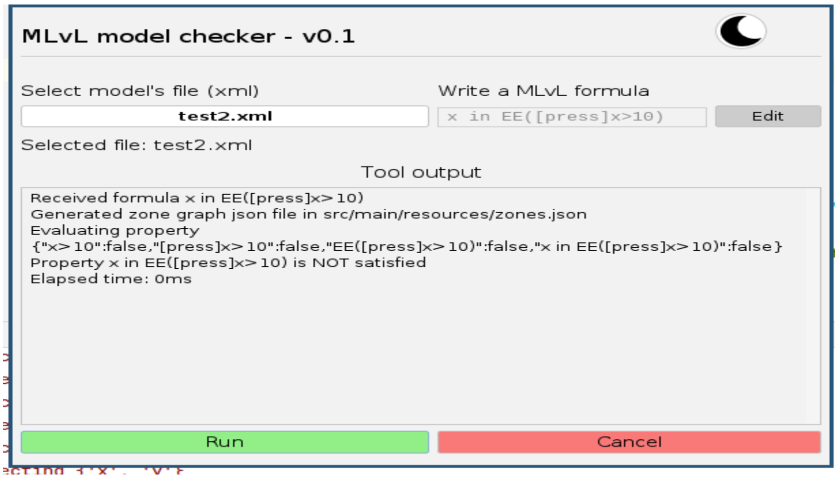
test2.xml

EE(y<=3 && y>=1 && <press>tt)

Selected file: test2.xml

```
Received formula x in EE(y<=3 && y>=1 && <press>tt)
Generated zone graph json file in src/main/resources/zones.json
Evaluating property
{"y<=3":true,"y>=1":true,"y<=3 && y>=1":true,"<press>tt":true,"EE(y<=3 && y>=1
&& <press>tt)":true,"x in EE(y<=3 && y>=1 && <press>tt)":true}
Property x in EE(y<=3 && y>=1 && <press>tt) is satisfied
Elapsed time: 0ms
```

Figure 8.20: MIMETIC Tool (Property is satisfied)



test2.xml

x in EE([press]x>10)

Selected file: test2.xml

```
Received formula x in EE([press]x>10)
Generated zone graph json file in src/main/resources/zones.json
Evaluating property
{"x>10":false,"[press]x>10":false,"EE([press]x>10)":false,"x in EE([press]x>10)":false}
Property x in EE([press]x>10) is NOT satisfied
Elapsed time: 0ms
```

Figure 8.21: MIMETIC Tool (Property is not satisfied)

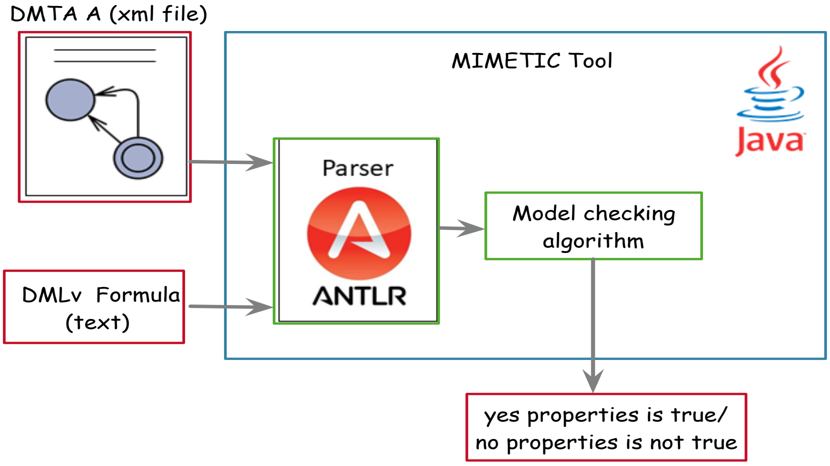


Figure 8.22: MIMETIC Tool

algorithm for the parallel composition of two DMTA, and a decision algorithm for multi-timed bisimulation using the (derivative) zone-based technique. The (derivative) multi-timed bisimulation algorithm is EXPTIME-complete. Another strength is the extension of ML_ν with rate constraints (DML_ν). The model check for the extended DML_ν formula interpreted over DMTA is EXPTIME-complete. The advanced DML_ν logic is sound and complete.

8.7.2 Weaknesses

Since DMTA is an extension of TA and MTA, it inherits all the weaknesses of TA and MTA (TA is neither determinizable nor complementable, and its inclusion problem is undecidable).

8.8 Wrap up

This chapter introduces Derivative Multi-timed Automata (DMTA), which are an extension of icTA and MTA, but with alternative semantics based on the concepts of independent clocks and rate constraints. In the second section, thanks to our definition of rate constraints, we were also able to extend the semantics of MTA (DMTA). We also extended the composition between two MTA (DMTA). In the third section, we have shown an EXPTIME algorithm for deciding whether two DMTA are multi-timed bisimilar. In the fourth section, we have extended the timed modal ML_ν logic with independent clocks and rate constraints. This gives us the multi-timed modal DML_ν, which we have shown to be PSPACE-complete. In the fifth section, we have shown some examples of distributed real-time models made over DMTA and DML_ν. In the sixth section, we have shown the functionalities and details related to the implementation of our algorithms. We have also presented some experimental

```
document        :  automataDeclr logicDeclr ;

automataDeclr   :   AUTOM ’:’ ID ’[’
                    clocksDeclr
                    locations
                    transitions
                    invariants
                    atomicProps
                    ’]’ ’;’
                ;
clocksDeclr     :   CLOCKS ’:’ ID (’,’ID)* ;
locations       :   LOCATIONS ’:’ ID (’,’ID)* ;
transitions     :   transition (’,’transition)* ;
transition      :   ID  ’-’ ID(’!’)? (’,’ clockConstraint)? ’,’ edge;
edge            :   (’{’ID’}’)? ’->’ ID ;
invariants      :   INVAR ’:’ ID SAT clockConstraint (’,’ ID SAT clockConstraint )
      ;
atomicProps     :   ATOMIC PROPOSITIONS ’:’ ID SAT proposition (’,’ ID SAT
     proposition)*
                ;
logicDeclr      :   DECL ID ’[’ ID SAT proposition ’]’ ’;’;

propositions    : proposition;

proposition     :   TOP                                         #tt
                |   BOT                                         #ff
                |   proposition (S|WS)? AND (S|WS)? proposition #conjunction
                |   proposition OR proposition                  #disjunction
                |   ’[’ID’]’ proposition                        #boxModality
                |   ’<’ID’>’ (S|WS)? proposition                #diamondModality
                |   ’EE’’(’ proposition ’)’                     #existsModality
                |   ’AA’’(’ proposition ’)’                     #forallModality
                |   clockFormula (S|WS)? ’in’ (S|WS)? proposition #inModality
                |   clockConstraint                             #clockCC
                |   clockFormula ’+’ NATURAL
                    comparisonOp clockFormula ’+’ NATURAL       #clockOperation
                |   rateConstraint                              #rateCC
                ;

clockConstraint :   clockFormula (S|WS)? comparisonOp (S|WS)? NATURAL
                |   clockConstraint (S|WS)? AND (S|WS)? clockConstraint
                |   TOP
                ;

rateConstraint  :   ID QUOTE (S|WS)? comparisonOp (S|WS)? NATURAL       #constantRC
                |   ID QUOTE (S|WS)? comparisonOp (S|WS)? ID QUOTE      #simpleRC
                |   rateConstraint (S|WS)? AND (S|WS)? rateConstraint   #
                     conjunctionRC
                |   TOP                                                 #trueRC
                |   BOT                                                 #falseRC
                ;

clockFormula:   ID (’^’ ID)?;
comparisonOp:   ’>’ | ’<’ | ’>=’ | ’<=’ |  EGA ;
```

Figure 8.23: Part of MIMETIC grammar

```
lexer grammar MLvLexer;


fragment
DIGIT   :   [0-9]   ;


QUIT        :   'quit'          ;
AUTOM       :   'Automata'      ;
DECL        :   'Declaration'   ;
FSYNCH      :   'Fsynchro'      ;
TABLE       :   'Table'         ;
SYSTEM      :   'System'        ;
INVAR       :   'Invariants'    ;
CLOCKS      :   'Clocks'        ;
LOCATIONS   :   'Locations'     ;
TRANSITIONS :   'Transitions'   ;
ATOMIC      :   'Atomic'        ;
PROPOSITIONS:   'Propositions'  ;
NULL        :   'NULL'  ;
ACTIVITIES  :   'Activity' | 'Activities';
IN          :   'in';
EXISTS      :   'EE';
FORALL      :   'AA';
AND         :   '&&' ;
OR          :   '||' ;
NOT         :   '-' ;
TOP         :   'tt';
BOT         :   'ff'    ;
ACCOG       :   '{' ;
ACCOD       :   '}' ;
CROG        :   '[' ;
CROD        :   ']' ;
PARG        :   '(' ;
PARD        :   ')' ;
AFF         :   ':' ;
FSEP        :   ';' ;
SEP         :   ',' ;
POINT       :   '.' ;
STAR        :   '*' ;
FLECHE      :   '->';
APPROX      :   '~' ;
SAT         :   '|=';
REC         :   '?' ;
SEN         :   '!' ;
PARAM       :   '%' ;
AROB        :   '@' ;
WITHOUT     :   '\\';
PLUS        :   '+' ;
EUNTIL      :   '[@)';
IMPL        :   '=>';
INFE        :   '<=';
INF         :   '<';
LESS        :   '&lt;';
SUPE        :   '>=';
SUP         :   '>';
EGA         :   '==';
SOUL        :   '_';
QUOTE       :   '\'';
NEGATIVE    :   '-'DIGIT+;
NATURAL     :   DIGIT+;
WEDGE       :   '^' ;
ID          :    [a-zA-Z_]([a-zA-Z0-9_])* ;
//COMMENT :   '#'(.)*?'\n' ;
WS      :   [ \t\n]+    ;
S       :   [ \t]+;
```

Figure 8.24: Part of MIMETIC Lexer

results. Finally, in the seventh section, we have shown the strengths and weaknesses of DMTA and DML_{ν}.

Part III

Postface

“Most of the fundamental ideas of science are essentially simple, and may, as a rule, be expressed in a language comprehensible to everyone.” Albert Einstein

CHAPTER 9

CONCLUSION AND FUTURE RESEARCH DIRECTIONS

A DRTS may be characterized by multiple communicating components (or processes) whose behavior depends on multiple timing constraints, and such components may reside on multiple computers distributed over a communication network. A DRTS may use synchronous clocks, i.e., all its components use the same clock, or asynchronous clocks, i.e., all its components have their independent clocks that are subject to clock drift [Cri96]. Synchronous and asynchronous models represent two forms in the modeling and implementation of DRTS. However, the majority of current implementations of DRTS combine the advantages of both models, which is known as timed asynchronous models. [Cri96]. In such a DRTS, each component has access to its local clock, which runs at a given rate of the global time. Components communicate with each other by passing messages, which can take an infinite amount of time to be transmitted [Cri96]. Formal verification techniques have been used to verify the logical correctness of DRTS with respect to its specification [Asp18].

In this thesis, we present three alternative semantics that can be used to model and specify the distributed behavior of the components of a DRTS, and such components have clocks that are not necessarily synchronized. We are also particularly interested in the correct operation of these DRTS.

9.1 Summary of contributions

Our contributions to the thesis have been on the following points: undecidability, perfect clock synchronization, complexity, single timestamps and large representation of states.

(i) In Chapter 6 we have proposed:

- An alternative (real-time) semantics based on independent (distributed) event clocks, allowing us to model and specify distributed and real-time properties. We have defined DECA and proved that they are fully decidable and that their language inclusion problem is PSPACE-complete, as for classical automata. We also showed that the universal (timed) languages of DECA are decidable and regular, in contrast to the universal languages of icTA. [ABG$^+$08].
- A logic interpreted on DECA (called DECTL) to specify distributed and real-time properties. We have shown that the problems of satisfiability, validity, and model checking are decidable for DECTL, more precisely PSPACE-complete, as for LTL.

(ii) In Chapter 7 we have proposed:

- An alternative semantics for capturing the execution of Multi-timed Automata (MTA) [OAS17], based on multi-timed words running over Multi-timed Labeled Transition Systems (MLTS). We extended the notion of timed bisimulation to such structures and proposed an EXPTIME algorithm for checking decidability.
- A logic interpreted on MTA (called ML_ν), which can accept multi-timed words and has an alternative semantics [OAS17]. ML_ν is an extension of the modal logic L_ν, tailored for alternative semantics over MTA. We have shown that the model checking problem over MTA is ExPTIME-complete, analogous to L_ν.

(iii) In Chapter 8 we have proposed:

- An alternative semantics based on the concepts of independent clocks and rate constraints. We extended the semantics of MTA to Multi-timed Automata with Clocks Derivatives (DMTA). We also gave an EXPTIME algorithm for deciding whether two DMTA are multi-timed bisimilar.
- A logic interpreted on DMTA (called DML_ν,) which can accept independent clocks and rate constraints. DML_ν is an extension of the modal logic ML_ν, tailored for alternative semantics over DMTA. We have shown that the model checking problem over DMTA is ExPTIME-complete, analogous to ML_ν and L_ν. We have shown the functionalities and details related to the implementation of our algorithms.
- The design and implementation of three tools (MUTES, MULTI-TEMPO, and MIMETIC) for simulating, modeling, and verifying DRTS. We have also presented some experimental results with our MUTES tool.

9.2 Perspectives and future work

The work presented in this thesis does not solve all the problems we currently face in the specification and verification of DRTS. This section The work presented in this thesis does not solve all the problems we currently face in the specification and verification of DRTS. This section presents our perspectives and future research directions related to our three alternative semantics and tools.

9.2.1 Distributed Event Clocks

We have proposed the basis of a theoretical framework for modeling and verifying distributed and RTS by introducing independent (or distributed) event clocks, inspired by [ABG$^+$08]. However, it is possible to extend DECA and DECTL with notions and representations from other formalisms:

(i) We can explore the possibility of extending DECA and DECTL with the notion of probability, which will allow us to model probability transitions in DECA and check whether DECTL properties hold for some acceptable probability. In our Probabilistic DECA (PDECA), a transition is enabled as long as the relevant guards and invariants hold, but the time to make an enabled transition depends on a certain probability distribution, as in Probabilistic Timed Automata (PTA) [JLS08].

(ii) We can explore the possibility of extending the known expressive equivalence of EventClockTL and MITL+Past [HRS98] to construct a distributed version of MITL (DMITL) with independent modalities $\mathcal{U}_I^p, \mathcal{S}_I^p$.

(iii) We can explore the possibility of extending a new first-order monadic logic with a metric quantifier $\exists t \in^p]\, t_0, t_0 + k [\, . \, \phi$, $\exists t \in^p]\, t_0 - k, t_0 [\, . \, \phi$, where ϕ has only the free variable t (see [HR06] for Quantitative Temporal Logic (QTL)). We can show the expressiveness of Distributed Quantitative Temporal Logic (DQTL).

(iv) We can explore the possibility of extending DECTL to allow not only the last ϕ but also the last n ϕ [OLS10]. This logic can still be translated into DECA.

(v) We can explore the possibility of adding DECA automata operators [Wol83].

(vi) We can explore the possibility of adding second-order quantification to predicates that are not subject to a real-time constraint.

(vii) We can explore the possibility of adding independent modalities $\mathcal{U}_I^p, \mathcal{S}_I^p$ in a linear μ calculus.

(viii) We can explore the possibility of extending DECA and DECTL with the notion of rate constraints and clock derivatives.

(ix) We can use DECA and DECTL to model and verify industrial applications.

(x) We can explore the possibility of implementing a model checking tool based on DECTL and DECA.

9.2.2 Multiple Independent Clocks

We have proposed a theoretical framework that allows timed bisimulation, L_ν and TA with independent clocks. However, it is possible to extend MTA and ML_ν with notions and representations from other formalisms:

(i) In the current decade, research in process calculi has sought to reduce the distance between formal models described by process calculi and DRTS. In particular, features such as time, mobility, and nondeterminism [D'A99] [OS06] [Sif01] [Ber04] [HR95] [SG11], have been introduced into the models. Some important examples of such process calculus for describing systems that require timing, mobility, and non-determinism are Timed CSP [SDJ$^+$92] [D'A99] [OS06], Timed CCS [Sif01] [HR95], Timed π-calculus (π_t-calculus

[Ber02] [Ber04] [SG11]), a widely studied formalism for describing and reasoning about DRTS and mobile processes. These formalisms are based on a combination of synchronous communication, parallel composition, choice, and real-time constructors. The characteristics of these calculi are that they consist of autonomous processes that communicate and operate on an absolute time reference that is available to all processes. Timed CSP [SDJ+92] [D'A99] [OS06], Timed CCS [Sif01] [HR95] and Timed π-Calculus [SG11] use elements of TA, such as clocks, timing constraints, which are usually more appropriate for describing and verifying RTS. We can explore the possibility of extending the π calculus with explicit notions of independent clocks. Our two contributions would be

  (a) To define the π_{mt}-calculus, a π-calculus with distributed real-time constructors and non-deterministic decisions. The π_{mt}-calculus will capture the idea of independent clocks, which is essential for compositionality of processes.
  (b) To define a relationship between π_{mt}-calculus and ML_ν, which is essential for the specification and verification of distributed and RTS.

(ii) We can explore the possibility of adding a least fixpoint operator to ML_ν.
(iii) We can explore the satisfiability of ML_ν by exploiting decidability results on Recursive Weighted Logic [LM14b], which has similar properties.
(iv) We would like to know what distributed real-time properties can be preserved on ML_ν.
(v) We can explore the possibility of parallelizing our multi-timed bisimulation algorithm.
(vi) We can explore the possibility of defining weak multi-timed bisimulation. We can also propose a multi-timed zone-based algorithm for checking weak multi-timed bisimulation.
(vii) We can explore the possibility of relating our approach to other formalisms mentioned in the introduction, such as HIOA, DRTL, APTL, Time Petri nets [GYH+22].

9.2.3 Distributed Clock Derivatives

We have proposed a theoretical framework that allows multi-timed bisimulation, ML_ν and MTA with clock rates and clock derivatives. However, it is possible to extend MTA and ML_ν with notions and representations from other formalisms:

(i) We can explore the possibility of relating DMTA to Hybrid Automata (HA) [Ras05].
(ii) We can explore the expressiveness of DMTA with respect to TA, icTA.
(iii) We can use DMTA and ML_ν to model and verify industrial applications.
(iv) We would like to know what distributed real-time properties can be preserved on ML_ν.
(v) We can add to our model checking tool MIMETIC the least and greatest fixpoint operators.

(vi) we can improve our MUTES and MIMETIC tools to investigate more general problems of verification, like performance evaluation.

(vii) We can explore the possibility of working on the implementability of DMTA to incorporate the platform information by explicitly modeling the execution platform [AT05].

Appendix A

ω-Automata

In this section, we give the omitted definitions in Section 4.2.

A.1 Determinization and Complementation of ω-automata

Complementing ω automata is a very complex problem. The reason is that the known complementation algorithms require deterministic automata, but we have already seen that deterministic Büchi automata (DBA) are not as expressive as non-deterministic ones. That is, for a given Non-Deterministic Büchi Automata (NBA), there may not even be a deterministic automaton, and thus no way to build the complementary automaton in a relatively easy way.

A.1.1 Determinization

The determinization operation consists in the transformation of a NBA into a DBA. However, any NBA cannot be determinized into an automaton with Büchi acceptance conditions.

Example 45. *Given a NBA $\mathscr{A}$ that recognizes an ω-word $(a+b)*\cdot b^{\omega}$. This notation means that a must be recognized only finitely often, but b infinitely often. This automaton is represented by Figure A.1*

Now suppose that we have a DBA $\mathscr{B} = (\{a,b\}, Q, \rightarrow_{DBA}, \{q_0\}, F)$ which recognizes the same language as $\mathscr{A}$, then: $u_0 = b^{\omega}$, then $u_0 \in \mathscr{L}(\mathscr{A})$ and there exists a finite prefix v_0 of u_0 that brings $\mathscr{B}$ to F, $u_1 = v_0ab^{\omega}$, then $u_1 \in \mathscr{L}(\mathscr{A})$ and there exists a finite prefix v_0av_1 of u_1 that brings $\mathscr{B}$ to F, $\cdots$, $u_n = v_{n-1}ab^{\omega}$, then $u_n \in \mathscr{L}(\mathscr{A})$ and there exists a finite prefix $v_0av_1a\cdots av_n$ of u_n that brings $\mathscr{B}$ to F. Since the number of states Q is finite, there exists i and j, $0 \leq i < j$ such as words $v_0av_1a\cdots av_i$ and

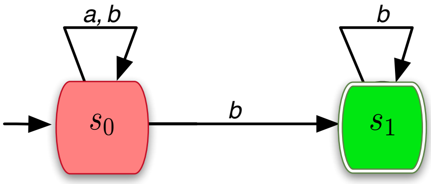


Figure A.1: A NBA.

$v_0 a v_1 a \cdots a v_j$ bring to the same state. Then $m = v_0 a v_1 a \cdots a v_i (a \cdots v_j)^\omega$ is accepted by $\mathscr{B}$. But since m has an infinity of a, it cannot be recognized by $\mathscr{A}$.

This example implies that to determinize a BA, the Büchi class cannot be used.

A.1.1.1 Deterministic Rabin Automata

Definition 111. *A deterministic Rabin automaton (DRA) is a tuple $\mathscr{A} = (Q, \Sigma, \rightarrow_{DRA}, q_0, F)$ such that*

- *Q is the finite set of locations.*
- *Σ is the finite alphabet, in our case $\Sigma = 2^{\mathbb{P}}$.*
- *$\rightarrow_{DRA}: Q \times \Sigma \rightarrow 2^Q$ is the transition relation.*
- *$q_0 \in Q$ is the initial location,*
- *$F = \{(L_1, U_1), \ldots, (L_r, U_r) | L_i, U_i \subseteq Q\}$ is the Rabin acceptance condition.*

Definition 112. *A run ρ of a Rabin automaton $\mathscr{A}$ with acceptance condition $F = \{(L_1, U_1), \ldots, (L_r, U_r)\}$ is accepting iff there exists an $i \in \{1, \ldots, r\}$ such that $inf(\rho) \cap L_i \neq \emptyset$ and $inf(\rho) \cap U_i = \emptyset$.*

In other words, an acceptance pair (L_i, U_i) is accepting when there is at least one location in L_i that is visited infinitely often in ρ and after some point no location in U_i is visited any more in ρ. If at least one of the r pairs (L_i, U_i) is accepting, then the whole automata is accepting.

A.1.1.2 Deterministic Muller Automata

Definition 113. *A deterministic Muller automaton (DMA) is a tuple $\mathscr{A} = (Q, \Sigma, \rightarrow_{DMA}, q_0, F)$ such that*

- *Q is the finite set of locations.*
- *Σ is the finite alphabet, in our case $\Sigma = 2^{\mathbb{P}}$.*
- *$\rightarrow_{DMA}: Q \times \Sigma \rightarrow 2^Q$ is the transition relation.*
- *$q_0 \in Q$ is the initial location,*
- *$F \subseteq 2^Q$ is the Muller acceptance condition.*

Definition 114. *A run ρ of Muller automaton $\mathscr{A}$ is accepting if the accepting set is included into the infinity set of visited locations. The formal definition for this acceptance condition is: $inf(\rho) \in F$, with $F \subseteq 2^Q$, the set of accepting states.*

Safra's construction [Saf89] produces a Muller or Rabin automaton from a NBA. For a NBA with n states, it creates a Deterministic Büchi Automata (DBA) with $2^{O(nlogn)}$ states, which is optimal for Rabin automata.

A.1.2 Complementation

Complementation for deterministic ω automata is a simple operation, since it only requires switching to the dual acceptance condition (Muller, Rabin, and Streett), but for nondeterministic ones this operation is more complicated. Safra [Saf89] proposed a construction by determinizing the automaton, which produces a Büchi automaton with $2^{O(nlogn)}$ states.

Appendix B

Abstractions and Bisimulation

However, there are also two other abstractions that can be found in literature, namely $\mathfrak{a}_{Extra^+_M}(\mathcal{Z})$ and $\mathfrak{a}_{Extra^+_{LU}}(\mathcal{Z})$ where for a clock zone $\mathcal{Z}$, $\mathcal{Z} \subseteq \mathfrak{a}_{Extra_M}(\mathcal{Z}) \subseteq \mathfrak{a}_{Extra^+_M}(\mathcal{Z})$ and $\mathcal{Z} \subseteq \mathfrak{a}_{Extra_{LU}}(\mathcal{Z}) \subseteq \mathfrak{a}_{Extra^+_{LU}}(\mathcal{Z})$) [BBLP04] [BBLP06]. The improvement of these two abstractions presented above is that if a whole clock zone is above the maximal bound for some clock x, then diagonal constraints involving x can be removed of the clock zones, even if they are not themselves above the maximum bound. These two abstractions presented above can also be defined in terms of extrapolation operators on DBMs, denoted by $Extra^+_M(\mathcal{D})$ and $Extra^+_{LU}(\mathcal{D})$ where D is the DBM in canonical form which represents the clock zone $\mathcal{Z}$. Formally, the extrapolation operator $\mathcal{D}' = Extra^+_M(\mathcal{D})$ with $\mathcal{D}' = (\mathbf{d}'_{i,j})_{0 \le i,j \le n}$ is defined as follows:

$$\mathbf{d}'_{i,j} = \begin{cases} (\infty, <) & if \quad d_{i,j} > M(x_i) \\ (\infty, <) & if \quad -d_{0,i} > M(x_i) \\ (\infty, <) & if \quad -d_{0,j} > M(x_j), i \neq 0 \\ (-M(x_j), <) & if \quad -d_{0,j} > M(x_j), i = 0 \\ (d_{i,j}, \prec_{i,j}) & otherwise \end{cases}$$

Similarly, the extrapolation operator $\mathcal{D}' = Extra^+_{LU}(\mathcal{D})$ with $\mathcal{D}' = (\mathbf{d}'_{i,j})_{0 \le i,j \le n}$ is defined as follows:

$$\mathbf{d}'_{i,j} = \begin{cases} (\infty, <) & if \quad d_{i,j} > L(x_i) \\ (\infty, <) & if \quad -d_{0,i} > L(x_i) \\ (\infty, <) & if \quad -d_{0,j} > U(x_j), i \neq 0 \\ (-U(x_j), <) & if \quad -d_{0,j} > U(x_j), i = 0 \\ (d_{i,j}, \prec_{i,j}) & otherwise \end{cases}$$

Algorithms B.1 and B.2 describe the $Extra^+_M$ and $Extra^+_{LU}$ operations on a DBM [BBFL03]. The algorithm B.1 works as follows: it repeatedly removes all upper bounds

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$ and $M(x_i)$  the maximal bound, where
    $x_{i_{0\leq i<n}}$ ∈ $X$.
Output: A DBM $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$.
DBM $D'$;
DBM ExtraMPlusOperator(DBM $\mathscr{D}$, List const $M$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       for(j=0; j<$\mathscr{D}$.size; j++){
         if(i ≠ j && ($\mathscr{D}$[i,j] > ($M(x_i$, <)) || $\mathscr{D}$[0,i] <
         ($-M(x_i$, <)) ||  $\mathscr{D}$[0,j] < ($-M(x_j$, <))) && $i \neq 0$) {
            $\mathscr{D}'$[i,j] = ($\infty$,<);
         }
        else if($\mathscr{D}$[i,j] < ($-M(x_j$), <) && $i = 0$) {
           $\mathscr{D}'$[i,j] = ($-M(x_j$), <);
        }
       }
     }
return $\mathscr{D}'$;
}
```

Algorithm B.1: $Extra^{+}_{M}$ Algorithm

```
Input: A DBM $\mathscr{D} = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$, $L(x_i)$ the maximal lower bound (and
    $U(x_i)$ the maximal upper bound), where $x_{i_{0\leq i<n}}$ ∈ $X$.
Output: A DBM $\mathscr{D}' = (\mathbf{d}_{i,j})_{0\leq i,j\leq n}$.
DBM $\mathscr{D}'$;
DBM ExtraLUPlusOperator(DBM $\mathscr{D}$, List const $L$, List const $U$) {
int i, j;
     for(i=0; i<$\mathscr{D}$.size; i++){
       for(j=0; j<$\mathscr{D}$.size; j++) {
         if(i ≠ j && ($\mathscr{D}$[i,j] > ($L(x_i$, <)) || $\mathscr{D}$[0,i] <
          ($-L(x_{(i)}$, <)) || $\mathscr{D}$[0,j] < ($-U(x_j$, <))) && $i \neq 0$) {
            $\mathscr{D}'$[i,j] = ($\infty$,<);
         }
        else if($\mathscr{D}$[i,j] < ($-U(x_j$), <)  && $i = 0$) {
           $\mathscr{D}'$[i,j] = ($-U(x_j$), <);
        }
       }
     }
return $\mathscr{D}'$;
}
```

Algorithm B.2: $Extra^{+}_{LU}$ Algorithm

higher than $M(x_i)$ with $x_i \in X$ and lowers all lower bounds higher than $M(x_i)$ with $x_i \in X$ down to the $M(x_i)$. The algorithm B.2 works as follows: it repeatedly removes all upper bounds higher than $U(x_i)$ with $x_i \in X$ and lowers all lower bounds higher than $L(x_i)$ with $x_i \in X$ down to the $L(x_i)$. Note that there operators do not preserve the canonical form of the DBM.

B.0.1 Time-abstract Bisimulation

Time-abstract bisimulation has been considered in [AD94], and is one of the fundamental concepts of region equivalence [AD94]. Time-abstract bisimulations are equivalences that abstract the quantitative aspects of time [TY01].

Definition 115 (**Time-Abstract Bisimulation** [TY01])**.** *Let $\mathscr{D}_1$ and $\mathscr{D}_2$ be two TLTS over the same set of actions Σ. Let $Q_{\mathscr{D}_1}$ (resp., $Q_{\mathscr{D}_2}$) be the set of states of $\mathscr{D}_1$ (resp., $\mathscr{D}_2$). A timed-abstract bisimulation over TLTS $\mathscr{D}_1, \mathscr{D}_2$ is a binary relation $\mathscr{R} \subseteq Q_{\mathscr{D}_1} \times Q_{\mathscr{D}_2}$ such that the following holds: if $q_{\mathscr{D}_1}\ \mathscr{R}\ q_{\mathscr{D}_2}$ then:*

(i) Discrete transition: For every $a \in \Sigma$ and for every $q_{\mathscr{D}_1} \xrightarrow{a}_{\mathscr{D}_1} q'_{\mathscr{D}_1}$, there exists a matching discrete transition $q_{\mathscr{D}_2} \xrightarrow{a}_{\mathscr{D}_2} q'_{\mathscr{D}_2}$ such that $q'_{\mathscr{D}_1} \mathscr{R} q'_{\mathscr{D}_2}$ and vice versa,

(ii) Delay transition: For every $d \in \mathbb{R}_{\geq 0}$ and for every $q_{\mathscr{D}_1} \xrightarrow{d}_{\mathscr{D}_1} q'_{\mathscr{D}_1}$, there exists a matching delay transition $q_{\mathscr{D}_2} \xrightarrow{d'}_{\mathscr{D}_2} q'_{\mathscr{D}_2}$ such that $q'_{\mathscr{D}_1} \mathscr{R} q'_{\mathscr{D}_2}$ and vice versa.

Two states $q_{\mathscr{D}_1}$ and $q_{\mathscr{D}_2}$ are timed-abstract bisimilar, written $q_{\mathscr{D}_1} \approx_{ma} q_{\mathscr{D}_2}$, iff there is a timed-abstract bisimulation $\mathscr{R}$ such that $q_{\mathscr{D}_1}\ \mathscr{R}\ q_{\mathscr{D}_2}$. $\mathscr{D}_1$ and $\mathscr{D}_2$ are timed-abstract bisimilar, written $\mathscr{D}_1 \approx_{ma} \mathscr{D}_2$, if there exists a timed bisimulation relation $\mathscr{R}$ over $\mathscr{D}_1$ and $\mathscr{D}_2$ containing the pair of initial states. Furthermore, for all $q_{\mathscr{D}_1}\ \mathscr{R}\ q_{\mathscr{D}_2}$, if $q_{\mathscr{D}_1} \in Q^F_{\mathscr{D}_1}$ then $q_{\mathscr{D}_2} \in Q^F_{\mathscr{D}_2}$.

APPENDIX C

DERIVATIVE MULTI-TIMED AUTOMATA

In this section, we give the omitted proof in Chapter 8.

C.1 Parallel Composition of DMTA

Now, we need to prove the parallel composition at the semantics for DMTA (e.g., $\mathsf{MLTS}(\mathscr{A},\tau)$ and $\mathsf{MLTS}(\mathscr{B},\tau)$).

Theorem 68. *Let $\mathscr{A}$ and $\mathscr{B}$ be two DMTA, then for any $\tau \in Rates$, $MLTS(\mathscr{A},\tau) \parallel MLTS(\mathscr{B},\tau) = MLTS((\mathscr{A} \parallel \mathscr{B}),\tau)$.*

Proof. The proof consists in showing that each transition of the $\mathsf{MLTS}(\mathscr{A},\tau) \parallel \mathsf{MLTS}(\mathscr{B},\tau)$ can be found in $\mathsf{MLTS}((\mathscr{A} \parallel \mathscr{B}),\tau)$ and vice versa. Let $\mathscr{R} = \{\, (((s_{\mathscr{A}},\, s_{\mathscr{B}}),\, \nu_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}),\, ((s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}}),(s_{\mathscr{B}},\nu_{\mathscr{B}},t_{\mathscr{B}}))) \mid \nu_{\mathscr{A}\parallel\mathscr{B}}(x) = \nu_{\mathscr{A}}(x) \uplus \nu_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$ and $t_{\mathscr{A}\parallel\mathscr{B}} = t_{\mathscr{A}} = t_{\mathscr{B}} \,\}$. Based on the MLTS and DMTA parallel composition, there exists two types of transitions:

(i) Discrete transition: Let $q_{\mathscr{A}} = (s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}},\nu_{\mathscr{B}},t_{\mathscr{B}})$ be two states of $\mathsf{MLTS}(\mathscr{A},\tau)$ and $\mathsf{MLTS}(\mathscr{B},\tau)$ respectively. A transition $((s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}}),(s_{\mathscr{B}},\nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}},\nu'_{\mathscr{A}},t'_{\mathscr{A}}),(s'_{\mathscr{B}},\nu'_{\mathscr{B}},t'_{\mathscr{B}}))$ exists on $\mathsf{MLTS}(\mathscr{A},\tau) \parallel \mathsf{MLTS}(\mathscr{B},\tau)$ iff the transition $((s_{\mathscr{A}},s_{\mathscr{B}}),\nu_{\mathscr{A}\parallel\mathscr{B}},\, t_{\mathscr{A}\parallel\mathscr{B}}) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}},s'_{\mathscr{B}}),\nu'_{\mathscr{A}\parallel\mathscr{B}},\, t'_{\mathscr{A}\parallel\mathscr{B}}))$ exists on $\mathsf{MLTS}((\mathscr{A},\tau) \parallel (\mathscr{B},\tau))$, with $\nu_{\mathscr{A}\parallel\mathscr{B}}$ and $\nu'_{\mathscr{A}\parallel\mathscr{B}}$ defined as $\nu_{\mathscr{A}\parallel\mathscr{B}}(x) = \nu_{\mathscr{A}}(x) \uplus \nu_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$, $\nu'_{\mathscr{A}\parallel\mathscr{B}} = \nu_{\mathscr{A}\parallel\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$:

(a) For $a \in \Sigma_{\mathscr{A}} \cap \Sigma_{\mathscr{B}}$: Let $((s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}}),\,(s_{\mathscr{B}},\nu_{\mathscr{B}},t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}},\nu'_{\mathscr{A}},t'_{\mathscr{A}}),\,(s'_{\mathscr{B}},\,\nu'_{\mathscr{B}},t'_{\mathscr{B}}))$ be a transition of $\mathsf{MLTS}(\mathscr{A},\tau) \parallel \mathsf{MLTS}(\mathscr{B},\tau)$. By Definition 78 (a), we have that the transition $((s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}}),(s_{\mathscr{B}},\nu_{\mathscr{B}},t_{\mathscr{B}})) \xrightarrow{a}_{mlts} ((s'_{\mathscr{A}},\,\nu'_{\mathscr{A}},\,t'_{\mathscr{A}}),\,(s'_{\mathscr{B}},\nu'_{\mathscr{B}},t'_{\mathscr{B}}))$ can not exist iff unless $(s_{\mathscr{A}},\nu_{\mathscr{A}},t_{\mathscr{A}}) \xrightarrow{a}_{mlts} (s'_{\mathscr{A}},\nu'_{\mathscr{A}},\,t'_{\mathscr{A}})$ in $\mathsf{MLTS}(\mathscr{A},\tau)$ and $(s_{\mathscr{B}},\nu_{\mathscr{B}},t_{\mathscr{B}}) \xrightarrow{a}_{mlts} (s'_{\mathscr{B}},\nu'_{\mathscr{B}},\,t'_{\mathscr{B}})$ in MLTS

$(\mathscr{B},\tau)$ both exist and the actions are synchronized. Therefore, since the two transitions on the MLTS, then we also know that there are two transitions on the corresponding DMTA, $s_{\mathscr{A}} \xrightarrow{a,\phi_{\mathscr{A}},Y_{\mathscr{A}}}_{\mathsf{dmta}} s'_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{a,\phi_{\mathscr{B}},Y_{\mathscr{B}}}_{\mathsf{dmta}} s'_{\mathscr{B}}$. Additionally, we have the clock valuations $v_{\mathscr{A}}$ and $v'_{\mathscr{A}}$ (and $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$ respectively) of each state defined respectively as $v'_{\mathscr{A}} = v_{\mathscr{A}}[Y_{\mathscr{A}} \leftarrow 0]$, $v_{\mathscr{A}} \models Inv(s'_{\mathscr{A}})$ and $v_{\mathscr{A}} \models R(s'_{\mathscr{A}})$ and similarly in order for $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$. Hence, the composition of $s_{\mathscr{A}} \xrightarrow{a,\phi_{\mathscr{A}},Y_{\mathscr{A}}}_{\mathsf{dmta}} s'_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{a,\phi_{\mathscr{B}},Y_{\mathscr{B}}}_{\mathsf{dmta}} s'_{\mathscr{B}}$ at the DMTA level is also based on the discrete transition of the DMTA composition. This lead to the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{a}_{\mathsf{dmta}} (s'_{\mathscr{A}}, s'_{\mathscr{B}})$. The clock valuation $v_{\mathscr{A}}$ (and $v_{\mathscr{B}}$) of each DMTA $\mathscr{A}$ (and $\mathscr{B}$) is projected on the result of their composition. By Definition 78, we have that the result of the composition of MLTS there exists three kinds of transitions and the clock valuation $v_{\mathscr{A}}$ (and $v_{\mathscr{B}}$) is is known for all kinds of transitions at the result of composition of DMTA, then we can generalize this fact to every transition $\rightarrow_{mlts}$ on the corresponding MLTS. The clock $v'_{\mathscr{A}}$ (and $v'_{\mathscr{B}}$) is projected into the synchronization of the composition of the DMTA. Then, the clock valuation that were reset by $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ will be reset by $v_{\mathscr{A}\|\mathscr{B}}$. Based on the composition of the DMTA, we know that the discrete transition that are enabled by $(s_{\mathscr{A}}, s'_{\mathscr{A}})$ (and $(s_{\mathscr{B}}, s'_{\mathscr{B}})$) will be enabled by $(s_{\mathscr{A}}, s'_{\mathscr{A}})$, $(s_{\mathscr{B}}, s'_{\mathscr{B}})$. This leads to $v'_{\mathscr{A}\|\mathscr{B}} = v_{\mathscr{A}\|\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$. Therefore, given the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{a}_{\mathsf{dmta}} (s'_{\mathscr{A}}, s'_{\mathscr{B}})$ and clock valuations $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$ then, we can obtain $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}), \xrightarrow{a}_{\mathsf{mlts}} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$.

Since the three cases hold we conclude that $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\mathsf{mlts}} ((q'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (q'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\mathsf{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{a}_{\mathsf{mlts}} ((q'_{\mathscr{A}}, q'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ implies $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\mathsf{mlts}} ((q'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (q'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$.

(ii) Delay transition: Let $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$ be two states of MLTS$(\mathscr{A},\tau)$ and MLTS$(\mathscr{B},\tau)$ respectively. A transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ exists on MLTS $(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$ iff the transition $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}}))$ exists on MLTS$((\mathscr{A},\tau) \parallel (\mathscr{B},\tau))$, with $v_{\mathscr{A}\|\mathscr{B}}$ and $v'_{\mathscr{A}\|\mathscr{B}}$ defined as $v_{\mathscr{A}\|\mathscr{B}}(x) = v_{\mathscr{A}}(x) \uplus v_{\mathscr{B}}(x)$ for $x \in X_{\mathscr{A}} \uplus X_{\mathscr{B}}$, $v'_{\mathscr{A}\|\mathscr{B}} = v_{\mathscr{A}\|\mathscr{B}}[(Y_{\mathscr{A}} \uplus Y_{\mathscr{B}}) \leftarrow 0]$:

(a) Let $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ be a transition of MLTS $(\mathscr{A},\tau) \parallel$ MLTS$(\mathscr{B},\tau)$. By Definition 78 (ii), we have that the transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}}), (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}}))$ can not exist iff unless $(s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}) \xrightarrow{\vec{d}}_{mlts} (s'_{\mathscr{A}}, v'_{\mathscr{A}}, t'_{\mathscr{A}})$ in MLTS $(\mathscr{A},\tau)$ and $(s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} (s'_{\mathscr{B}}, v'_{\mathscr{B}}, t'_{\mathscr{B}})$ in MLTS$(\mathscr{B},\tau)$ both exist. Therefore, since the two transitions on the MLTS, then we also know that there are two transitions on the corresponding DMTA $q_{\mathscr{A}} \xrightarrow{\vec{d}}_{\text{dmta}} q^{'}_{\mathscr{A}}$ and $q_{\mathscr{B}} \xrightarrow{\vec{d}}_{\text{mta}} q^{'}_{\mathscr{B}}$ where $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$. Additionally, we have the clock valuations $v_{\mathscr{A}}$ and $v'_{\mathscr{A}}$ (and $v_{\mathscr{B}}$ and $v'_{\mathscr{B}}$ respectively) of each state defined respectively as $\vec{d} = \tau(t^{'}_{\mathscr{A}}) - \tau(t_{\mathscr{A}})$ and $\forall\, t \in [t_{\mathscr{A}}, t^{'}_{\mathscr{A}}] : v + (\tau(t) - \tau(t_{\mathscr{A}})) \models \mathit{Inv}(s_{\mathscr{A}})$ and $v + (\tau(t) - \tau(t_{\mathscr{A}})) \models R(s_{\mathscr{A}})$ and similarly in order for $v_{\mathscr{B}}$ and and $v^{'}_{\mathscr{B}}$. Hence, the composition of $s_{\mathscr{A}} \xrightarrow{\vec{d}}_{\text{dmta}} s^{'}_{\mathscr{A}}$ and $s_{\mathscr{B}} \xrightarrow{\vec{d}}_{\text{dmta}} s^{'}_{\mathscr{B}}$ at the DMTA level is also based on the delay transition of the DMTA composition. This lead to the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{dmta}} (s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}})$. The clock valuation $v_{\mathscr{A}\parallel\mathscr{B}}$ of each DMTA is projected on the result of their composition. Then, the clock valuation that were reset by $v_{\mathscr{A}}$ and $v_{\mathscr{B}}$ will be reset by $v_{\mathscr{A}\parallel\mathscr{B}}$. Based on the composition of the DMTA, we know that the delay transition that are enabled by $(s_{\mathscr{A}}, s^{'}_{\mathscr{A}})$ and $(s_{\mathscr{B}}, s^{'}_{\mathscr{B}})$ will be enabled by $(s_{\mathscr{A}}, s_{\mathscr{B}}), (s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}})$. Therefore, given the transition $(s_{\mathscr{A}}, s_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{dmta}} (s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}})$ and clock valuations $v_{\mathscr{A}\parallel\mathscr{B}}$ and $v'_{\mathscr{A}\parallel\mathscr{B}}$ then, we can obtain $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}), \xrightarrow{\vec{d}}_{\text{mlts}} ((s^{'}_{\mathscr{A}}, s^{'}_{\mathscr{B}}), v'_{\mathscr{A}\parallel\mathscr{B}}, t'_{\mathscr{A}\parallel\mathscr{B}})$.

(b) Let $((s_{\mathscr{A}}, s_{\mathscr{B}}), v_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}) \xrightarrow{\vec{d}}_{mlts} ((s'_{\mathscr{A}}, s'_{\mathscr{B}}), v'_{\mathscr{A}\parallel\mathscr{B}}, t'_{\mathscr{A}\parallel\mathscr{B}}))$ be a transition of $\mathsf{MLTS}(\mathscr{A} \parallel \mathscr{B}, \tau)$. By Definition 78 (ii), we have that the transition $(q_{\mathscr{A}}, q_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} (q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}})$ and the clock valuation $v_{\mathscr{A}\parallel\mathscr{B}}$ and $v'_{\mathscr{A}\parallel\mathscr{B}}$. Therefore, since the two transitions on the composition of $\mathscr{A}$ and $\mathscr{B}$, then by the discrete transition of the composition of the two DMTA, we know that $q_{\mathscr{A}} \xrightarrow{\vec{d}}_{\text{dmta}} q^{'}_{\mathscr{A}}$ and $q_{\mathscr{B}} \xrightarrow{\vec{d}}_{\text{dmta}} q^{'}_{\mathscr{B}}$ both transition exist and $q_{\mathscr{A}} = (s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}})$ and $q_{\mathscr{B}} = (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})$. Based on the composition of the clock valuation $v'_{\mathscr{A}\parallel\mathscr{B}}$ is defined respectively as $\vec{d} = \tau(t^{'}_{\mathscr{A}\parallel\mathscr{B}}) - \tau(t_{\mathscr{A}\parallel\mathscr{B}})$ and $\forall\, t \in [t_{\mathscr{A}\parallel\mathscr{B}}, t^{'}_{\mathscr{A}\parallel\mathscr{B}}] : v_{\mathscr{A}\parallel\mathscr{B}} + (\tau(t) - \tau(t_{\mathscr{A}\parallel\mathscr{B}})) \models \mathit{Inv}_{\mathscr{A}}(s_{\mathscr{A}}) \wedge \mathit{Inv}_{\mathscr{B}}(s_{\mathscr{B}})$ and $(\tau, t) \models R_{\mathscr{A}}(s_{\mathscr{A}}) \wedge R_{\mathscr{B}}(s_{\mathscr{B}})$ and similarly in order for $v_{\mathscr{B}}$ and $v^{'}_{\mathscr{B}}$. Hence, the two transitions at the DMTA level along with the clock valuation of $v_{\mathscr{A}\parallel\mathscr{B}}$ and $v'_{\mathscr{A}\parallel\mathscr{B}}$, is applied which leads into $(s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}) \xrightarrow{\vec{d}}_{\text{dmta}} (s^{'}_{\mathscr{A}}, v^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}})$ of $\mathsf{MLTS}((\mathscr{A}, \tau)$ and $(s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{ta}} (s^{'}_{\mathscr{B}}, v^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}})$ of MLTS $((\mathscr{B}, \tau)$. Now the composition of these two transitions at the MLTS level is also based on the of discrete transition the DMTA composition. This leads into the transition $((s_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (s_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((s^{'}_{\mathscr{A}}, v^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (s^{'}_{\mathscr{B}}, v^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$ which happens to be our awaited conclusion.

Since the two implications hold we conclude that:

- Discrete transition: $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, v_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, v^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, v^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}) \xrightarrow{a}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), v'_{\mathscr{A}\parallel\mathscr{B}}, t'_{\mathscr{A}\parallel\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), v_{\mathscr{A}\parallel\mathscr{B}}, t_{\mathscr{A}\parallel\mathscr{B}}) \xrightarrow{a}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), v'_{\mathscr{A}\parallel\mathscr{B}}, t'_{\mathscr{A}\parallel\mathscr{B}})$ implies $((q_{\mathscr{A}}, v_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}},$

$\nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{a}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, \nu^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, \nu^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$.

- Delay transition: $((q_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, \nu^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, \nu^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$ implies $((q_{\mathscr{A}}, q_{\mathscr{B}}), \nu_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), \nu'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ and $((q_{\mathscr{A}}, q_{\mathscr{B}}), \nu_{\mathscr{A}\|\mathscr{B}}, t_{\mathscr{A}\|\mathscr{B}}) \xrightarrow{\vec{d}}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, q^{'}_{\mathscr{B}}), \nu'_{\mathscr{A}\|\mathscr{B}}, t'_{\mathscr{A}\|\mathscr{B}})$ implies $((q_{\mathscr{A}}, \nu_{\mathscr{A}}, t_{\mathscr{A}}), (q_{\mathscr{B}}, \nu_{\mathscr{B}}, t_{\mathscr{B}})) \xrightarrow{\vec{d}}_{\text{mlts}} ((q^{'}_{\mathscr{A}}, \nu^{'}_{\mathscr{A}}, t^{'}_{\mathscr{A}}), (q^{'}_{\mathscr{B}}, \nu^{'}_{\mathscr{B}}, t^{'}_{\mathscr{B}}))$.

Since (1), (2) and (3), we conclude that for any $\tau \in Rates$, $\mathsf{MLTS}(\mathscr{A}, \tau) \parallel \mathsf{MLTS}(\mathscr{B}, \tau) \approx \mathsf{MLTS}(\mathscr{A} \parallel \mathscr{B}, \tau)$.

□

proposition 31. *Let $\mathcal{Z}$ be a clock zones. Then, $\mathcal{Z} \uparrow_{\psi}$ and $\mathcal{Z} \downarrow_{\psi}$ are also clock zones.*

Proof. Let $\mathcal{Z}$ be zones, then we need to prove the following operations are clock zones:

(i) $\mathcal{Z} \uparrow_{\psi}$,

(ii) $\mathcal{Z} \downarrow_{\psi}$,

(i) Let $\mathcal{Z}$ be a clock zone and X be a set of clocks. We are going to demonstrate the fifth case (5) by showing that if $\mathcal{Z}$ is a clock zone, then a clock valuation $\nu' \in \mathcal{Z} \uparrow_{\psi}$, if there exists a tuple $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$, a tuple $\tau \in Rates$, a rate constraint $\psi \in \Psi(X)$, $t, t' \geq 0$, $t \leq t'$ with $\vec{d} = \tau(t') - \tau(t)$ such that if $(\tau, t) \models \psi$, then there exists a clock valuation $\nu \in \mathcal{Z}$ such that $\nu + \vec{d} = \nu'$. Thus, $\mathcal{Z} \uparrow_{\psi}$ is a clock zone. In order to demonstrate this, we need to solving the following system of inequalities:

$$\begin{cases} -c_{0,i} - \nu(x_i) \sim \vec{d} & for\ all\ x_i \in X \\ \vec{d} \sim c_{i,0} - \nu(x_i) & for\ all\ x_i \in X \\ \nu(x_i + \vec{d}) - \nu(x_j + \vec{d}) \sim c_{i,j} & for\ all\ x_i, x_j \in X \\ \vec{d} \geq 0 \end{cases}$$

(a) Let x_i be an independent clock, such that $x_i \in X$. For all $\nu \in \mathcal{Z}$, we know that if -$c_{0,i}$ - $\nu(x_i) \sim \vec{d}$ then -$c_{0,i} \sim \nu'(x_i)$ from which we can deduce that the inequality does not force $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ to be negative. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). This leads us to that for all $x_i \in X$, exists t, $t' \geq 0$, $t \leq t'$, such that, $(\tau, t) \models \dot{x}_i \sim 1$ iff τ_{x_i} is derivable at t and $d\tau_{x_i}/dt(t) \sim 1$. Let $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ be such solution. We let ν be the valuation such that $\nu(x_i) = \nu'(x_i) - \vec{d}$ for all $x_i \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $(\tau, t) \models \dot{x}_i \sim 1$ and $\nu'(x_i) = \nu(x_i) + \vec{d}$ with $\nu \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $\nu' \in \mathcal{Z} \uparrow_{\psi}$ and $\mathcal{Z} \uparrow_{\psi}$ is a clock zone.

(b) Let x_i be an independent clock such that $x_i \in X$. For all $\nu \in \mathcal{Z}$, we know that if $\vec{d} \sim c_{i,0} - \nu(x_i)$ then $\nu'(x_i) \sim c_{i,0}$ from which we can deduce that the inequality does not force $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ to be negative. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). This leads us to that for all $x_i \in X$, exists t, $t' \geq 0$,

$t \leq t'$, such that, $(\tau, t) \models \dot{x}_i \sim 1$ iff τ_{x_i} is derivable at t and $d\tau_{x_i}/dt(t) \sim$ 1. Let $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ be such solution. We let v be the valuation such that $v(x_i) = v'(x_i) - \vec{d}$ for all $x_i \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $(\tau, t) \models \dot{x}_i \sim 1$ and $v'(x_i) = v(x_i) + \vec{d}$ with $v \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $v' \in \mathcal{Z} \uparrow_{\psi}$ and $\mathcal{Z} \uparrow_{\psi}$ is a clock zone.

(c) Let x_i, x_j be two independent clocks such that x_i, $x_j \in X$. For all $v \in \mathcal{Z}$, we know that $(v(x_i) + \vec{d})$ - $(v(x_j) + \vec{d}) \sim c_{i,j}$, but, due to the fact that the clocks evolve at rates that can be independent of each other, clock differences can change over time and the two occurrences of $\vec{d}$ do not cancel each other out, then we can deduce that the inequality $v'(x_i)$ - $v'(x_j) \sim c_{i,j}$ is already in the appropriate form. Then the set of solutions of the inequality is not empty (i.e., the inequality is pairwise coherent). This leads us to that for all x_i, $x_j \in X$, exists t, $t' \geq 0$, $t \leq t'$, such that $(\tau, t) \models \dot{x}_i \sim \dot{x}_j$ iff τ_{x_i} is derivable at t and τ_{x_j} is derivable at t and $d\tau_{x_i}/dt(t) \sim d\tau_{x_j}/dt(t)$. Let $\vec{d} \in \mathbb{R}^{Proc}_{\geq 0}$ be such solution. We let v be the valuation such that $v(x_i) = v'(x_i) - \vec{d}$ and $v(x_j) = v'(x_j) - \vec{d}$ for all x_i, $x_j \in X$. Such a valuation exists, and is in $\mathcal{Z}$ by construction. Then, since $(\tau, t) \models \dot{x}_i \sim \dot{x}_j$ and $v'(x_i) = v(x_i) + \vec{d}$ and $v'(x_j) = v(x_j) + \vec{d}$ with $v \in \mathcal{Z}$ and some $\vec{d} \in \mathbb{R}^{X}_{\geq 0}$ and $t \in \mathbb{R}_{\geq 0}$ we can deduce that $v' \in \mathcal{Z} \uparrow_{\psi}$ and $\mathcal{Z} \uparrow_{\psi}$ is a clock zone.

(ii) $\mathcal{Z} \downarrow_{\psi}$. The argument is symmetric to (5).

□

Definition 116 (**Discrete Successor**). *Let $q = (s, \mathcal{Z})$ be a zone and e=$(s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of $\mathscr{A}$, then post$(\mathcal{Z}, e)$=$\{v' \mid \exists v \in \mathcal{Z}, \exists \tau \in Rates, (s, v) \xrightarrow{e}_{mlts(\mathscr{A},\tau)} (s', v')\}$ is the set of valuations that q can reach by taking the transition e.*

Intuitively, the zone $(s', \mathsf{post}(\mathcal{Z}, e))$ describes the discrete successor of the zone $(s, \mathcal{Z})$ under the transition e.

Definition 117 (**Discrete Predecessor**). *Let $q = (s', \mathcal{Z}')$ be a zone and e=$(s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of $\mathscr{A}$, then pred$(\mathcal{Z}', e)$=$\{v \mid \exists v' \in \mathcal{Z}', \exists \tau \in Rates, (s, v) \xrightarrow{e}_{mlts(\mathscr{A},\tau)} (s', v')\}$ is the set of valuations that q can reach by executing the transition e.*

The zone $(s, \mathsf{pred}(\mathcal{Z}', e)) \downarrow_{\psi}$ describes the discrete predecessor of the zone $(s', \mathcal{Z}')$ under the transition e. The set $\mathsf{pred}(\mathcal{Z}', e) \uparrow_{\psi}$ can be computed using the operations inverse clock reset and intersection on clock zones as follows:

$$\mathsf{pred}(\mathcal{Z}', e) \downarrow_{\psi} = ((\mathcal{Z}' \uparrow_{Y} \cap \phi) \cap Inv(s')).$$

The set $\mathsf{post}(\mathcal{Z}, e) \uparrow_{\psi}$ can be obtained using the operations clock reset and the standard intersection on clock zones as follows:

$$\mathsf{post}(\mathcal{Z}, e) \uparrow_{\psi} = ((\mathcal{Z} \cap (\phi \cap Inv(s'))) \downarrow_{Y} \cap Inv(s')).$$

The sets post$(\mathcal{Z}, e) \uparrow_{\psi}$ and pred$(\mathcal{Z}', e)$ are thus also clock zones, since they are a combination of zones preserving operators from Lemma 16.

proposition 32. *Let $\mathcal{A}$ be a DMTA, $e = (s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of a DMTA $\mathcal{A}$ and $(s, \mathcal{Z})$ be a zone, then post$(\mathcal{Z}, e) \uparrow_{\psi} = ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_X \cap Inv(s'))$ and pred$(\mathcal{Z}', e) \downarrow_{\psi} = ((\mathcal{Z}' \uparrow_X \cap \phi) \cap Inv(s))$.*

Proof. Let $(s, \mathcal{Z})$ be a zone and $e = (s, a, \phi, Y, s') \in \rightarrow_{dmta}$ be a transition of an DMTA $\mathcal{A}$, then we need to prove the following equalities:

(i) post$(\mathcal{Z}, e) \uparrow_{\psi} = ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_Y \cap Inv(s'))$,

(ii) pred$(\mathcal{Z}', e) \downarrow_{\psi} = ((\mathcal{Z}' \uparrow_Y \cap \phi) \cap Inv(s))$.

(i) Let $\mathcal{Z}$ be a convex clock zone and Y be a set of clocks. We are going to prove the third case (3) by showing that every clock valuation that is in post$(\mathcal{Z}, e) \uparrow_{\psi}$ is also in $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$ and vice versa (i.e., post$(\mathcal{Z}, e) \uparrow_{\psi} \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$ and $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s')) \subseteq$ post$(\mathcal{Z}, e) \uparrow_{\psi}$.

(a) post$(\mathcal{Z}, e) \uparrow_{\psi} \subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_Y \cap Inv(s'))$: Consider an arbitrary clock valuation v'. Then, we assume $v' \in$ post$(\mathcal{Z}, e) \uparrow_{\psi}$ and show $v' \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$. Since $v' \in$ post$(\mathcal{Z}, e) \uparrow_{\psi}$, then, $\exists(\tau, t) \models R(s')$ and exists $t \in \mathbb{R}_{\geq 0}$ such that $(s, v, t) \xrightarrow{e}_{mlts} (s', v', t)$. By Definition 98(1) there exists a discrete transition between s and s' with $v \models \phi$ and $v' = v[Y \leftarrow 0]$ and $v' \models Inv(s')$ and $(\tau, t) \models R(s')$. By Definition 43(3) it follows that $(\tau, t) \models R(s')$. Therefore, by intersection of zones and Definition 103(1), we have that $v' \in (\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y$ and $v' \in Inv(s')$ then $v' \in (\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap v' \in Inv(s')$. Thus, we have post$(\mathcal{Z}, e) \uparrow_{\psi}$ $\subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$ is a zone.

(b) $((\mathcal{Z} \cap (\phi \cap Inv(s))) \downarrow_Y \cap Inv(s')) \subseteq$ post$(\mathcal{Z}, e) \uparrow_{\psi}$: Consider an arbitrary clock valuation v. $v' \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$ and show v' $\in$ post$(\mathcal{Z}, e) \uparrow_{\psi}$. Since $v' \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y \cap Inv(s'))$, then, $v' \in ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_Y$ and $v' \in Inv(s')$. By Definition 103(3) and conjunction of zones, we have that $v' \in ((\mathcal{Z} \cap (\phi \cap Inv(s))$ and $v' \in Inv(s')$, then $v' \in (\mathcal{Z} \cap (\phi \cap Inv(s))) \cap Inv(s') \subseteq$ post$(\mathcal{Z}, e) \uparrow_{\psi}$. This is exactly the definition of inclusion of zones $v \in$ post$(\mathcal{Z}, e) \uparrow_{\psi}$. Thus, we have post$(\mathcal{Z}, e) \uparrow_{\psi}$ $\subseteq ((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$ is a zone.

The proof of these zones establishes the desired equality post$(\mathcal{Z}, e) \uparrow_{\psi} =$ $((\mathcal{Z} \cap (\phi \cap Inv(s)) \downarrow_X \cap Inv(s'))$.

(ii) pred$(\mathcal{Z}', e) \downarrow_{\psi} = ((\mathcal{Z}' \uparrow_Y \cap \phi) \cap Inv(s))$. The argument is symmetric to (3).

□

proposition 33 (Completeness). *Let $\theta = (s_0, v_0, t_0) \xrightarrow{\vec{d}_0, a_0} (s_1, v_1, t_1) \xrightarrow{\vec{d}_1, a_1} (s_2, v_2, t_2) \ldots \xrightarrow{\vec{d}_{n-1}, a_{n-1}} (s_n, v_n, t_n)$ be an initial (but not necessarily accepting) run of MLTS$(\mathcal{A}, \tau)$, for some $\tau \in Rates$. Then, for any state (s_i, v_i, t_i), where $0 \leq i \leq n$, appearing in this run, there exists a symbolic zone $(s_i, \mathcal{Z}_i)$ added in Q such that $v_i \in \mathcal{Z}_i$.*

Proof. We proceed by induction on the length of the run leading to (s_i, v_i, t_i).
Base case: We know that $v_0 \in \mathcal{Z}_0$. The zone $(s_0, Extra^{+}_{LU(s_0)}(\mathcal{Z}_0))$ is added to D and

Q in line 9.
Induction case: Assume that for all $0 \leq i \leq m$, there exists $(s_i, \mathcal{Z}_i)$ in Q such that $v_i \in \mathcal{Z}_i$. We will now show that there exists $(s_{m+1}, \mathcal{Z}_{m+1})$ in Q such that $v_{m+1} \in \mathcal{Z}_{m+1}$. By the induction hypothesis, we have $(s_m, \mathcal{Z}_m)$ in Q such that $v_m \in \mathcal{Z}_m$. Consider the transition $(s_m, v_m, t_m) \xrightarrow{\vec{d}_m, a_m} (s_{m+1}, v_{m+1}, t_{m+1})$ of the run θ and let e_m be its transition. As $(s_m, \mathcal{Z}_m)$ is in Q, the delay transition $\xrightarrow{\epsilon}_{\mathsf{ZG}}$ has been considered in the line 25 and represents $d_m > 0$. The other case $\vec{d}_m = 0$ means that $v' = v$ and is thus already included in $\mathcal{Z}_m$. Let $v'_m = v_m + \vec{d}_m$. Then, since $(s_m, v_m, t_m) \xrightarrow{d_m} (s_m, v'_m, t'_m)$ is a delay transition of the $\mathsf{MLTS}(\mathscr{A}, \tau)$, we have the time successor (i.e., $\mathcal{Z}_m = \mathcal{Z}_2 \leftarrow Extra^+_{LU}((\mathcal{Z}_1 \uparrow_{R(s)}) \wedge I(s)))$ and thus at line 27 we have added $\mathcal{Z}'_m = \mathcal{Z}_3$ or $\mathcal{Z}'_m = \mathcal{Z}_2$ at line 30, but in any case $v'_m \in \mathcal{Z}_2 \subseteq Extra^+_{LU}(\mathcal{Z}_2) \subseteq \mathcal{Z}_3$. Let $(s_m, \mathcal{Z}'_m) \xrightarrow{e_m}_{\mathsf{ZG}} (s_{m+1}, \mathcal{Z}_{m+1})$ be the discrete transition in the zone graph in lines 14 and 17. Let $(s_m, \mathcal{Z}'_m) \xrightarrow{e_m}_{\mathsf{ZG}} (s_{m+1}, \mathcal{Z}_{m+1})$ be the transition in the zone graph in lines 26 and 27. By definition of the symbolic transition, $v_{m+1} \in \mathcal{Z}_{m+1}$. If $(s_{m+1}, \mathcal{Z}_{m+1})$ is in Q, we are done. The only other case when $(s_{m+1}, \mathcal{Z}_{m+1})$ is not in Q is when there exists $(s_{m+1}, \mathcal{Z}'_{m+1})$ in Q such that $\mathcal{Z}_{m+1} \subseteq \mathcal{Z}'_{m+1}$. Therefore, $v_{m+1} \in \mathcal{Z}_{m+1}$ and since $(s_{m+1}, \mathcal{Z}'_{m+1})$ is in Q, our required zone would be $(s_{m+1}, \mathcal{Z}'_{m+1})$. □ □

proposition 34. *Let $q = (s, \mathcal{Z})$, $q' = (s, \mathcal{Z}') \in Q$ be two zones, then $\mathsf{TimePred}_{\uparrow_\psi}$ $(\mathcal{Z}, \mathcal{Z}')$ is a clock zone.*

The proof follows the same lines as the proof of [TY01] (p.48).

Proof. Let $q = (s, \mathcal{Z})$, $q' = (s, \mathcal{Z}')$ and $\mathcal{Z}'' = \mathsf{TimePred}_{\uparrow_\psi}(\mathcal{Z}, \mathcal{Z}')$. We show that $\mathcal{Z}''$ is convex, i.e, if v_1, $v_2 \in \mathcal{Z}''$ then $v = kv_1 + (1-k)v_2 \in \mathcal{Z}''$, for $0 < k < 1$. v_1, $v_2 \in \mathcal{Z}''$ implies that v_1, $v_2 \in \mathcal{Z}$ and $\exists\ \vec{d}_1, \vec{d}_2 \in \mathbb{R}^{Proc}_{>0}$ such that $v_1 + \vec{d}_1$, $v_2 + \vec{d}_2 \in \mathcal{Z}'$ and $\exists\ t_1$, $t_2 > 0$, $t_1 \leq t_2$, $\forall\ t'$, $t_1 \leq t' \leq t_2$, $\vec{d}_1 = \tau(t_2) - \tau(t)$ and $\vec{d}_2 = \tau(t') - \tau(t_1)$ then $v_1 + \vec{d}_1 \in \mathcal{Z}'$, $v_2 + \vec{d}_2 \in (\mathcal{Z} \cup \mathcal{Z}')$ and $\exists\ \psi \in \Psi(X)$, $(\tau, t') \models \psi$. Let $\vec{d} = k\vec{d}_1 + (1-k)\vec{d}_2$, then $v + \vec{d} = k(v_1 + \vec{d}_1) + (1-k)(v_2 + \vec{d}_2)$, implying that $v + \vec{d} \in \mathcal{Z}'$, since $\mathcal{Z}'$ is a clock zone. Now, we have to show that $\forall\ t'$, $t_1 \leq t' \leq t_2$, $\vec{d} = \tau(t_2) - \tau(t')$ and $\vec{d}' = \tau(t') - \tau(t_1)$, $v + \vec{d}' \in (\mathcal{Z} \cup \mathcal{Z}')$ and $\exists\ \psi \in \Psi(X)$, $(\tau, t') \models \psi$. Given $\vec{d}'$, $\vec{d}$, we can write $\vec{d}'$ as $k\vec{d}_3 + (1-k)\vec{d}_4$, for some $\vec{d}_3$, $\vec{d}_1$ and $\vec{d}_4$, $\vec{d}_2$. We have $v_1 + \vec{d}_3$, $v_2 + \vec{d}_4 \in (\mathcal{Z} \cup \mathcal{Z}')$. If both $v_1 + \vec{d}_3$, $v_2 + \vec{d}_4 \in \mathcal{Z}$ or $v_1 + \vec{d}_3$, $v_2 + \vec{d}_4 \in \mathcal{Z}'$, we are done, since $\mathcal{Z}$ and $\mathcal{Z}'$ are both clock zones. Considerer the case $v_1 + \vec{d}_3 \in \mathcal{Z}$ and $v_2 + \vec{d}_4 \in \mathcal{Z}'$. Let g be the smallest positive real such that $v_2 + \vec{d}_4 - g \in \mathcal{Z}$ or $v_1 + \vec{d}_3 - g(1 - \frac{1}{k}) \in \mathcal{Z}'$. To assume the first case, we have $v_1 + \vec{d}_5$, $v_2 + \vec{d}_6 \in \mathcal{Z}$, for $\vec{d}_6 = \vec{d}_4 - g$ and $\vec{d}_5 = \vec{d}_3 - g(1 - \frac{1}{k})$. Moreover, $\vec{d}' = k\vec{d}_5 + (1-k)\vec{d}_6$, which means that $v + \vec{d}' = k(v_1 + \vec{d}_5) + (1-k)(v_2 + \vec{d}_6)$. By convexity of $\mathcal{Z}$, $v + \vec{d}' \in \mathcal{Z}$. □

proposition 35. *Let $(s, \mathcal{Z})$ be a zone of Π and let $(e_{\mathscr{A}}, e_{\mathscr{B}})$ be an edge of the $\mathsf{DMZG}(\mathscr{C})$, then each of $\mathsf{TimeRefine}(\mathcal{Z}, \Pi)$ and $\mathsf{DiscreteSigRefine}(\mathcal{Z}, \Pi)$ forms a set of convex zones $\mathcal{Z}$ in Π.*

Proof. (i) Consider $\Pi_1 = \mathsf{TimeRefine}(\mathcal{Z}, \Pi)$. By Lemma 27, all members of Π are zones. It remains to show that their union yields $\mathcal{Z}$. Let $\mathcal{Z}_i \in \Pi_1$, $\mathcal{Z}_i =$

```
Input: A DMTA $\mathscr{C}$=($S, s_0, \Sigma, X, \rightarrow_{ta}, Inv, F, \pi$)
Output: A reachable zone graph DMZG($\mathscr{C}$) = $(Q, q_0, (\Sigma \cup \{\epsilon\}), T_{ZG})$
//$s \in S$ is a location of $\mathscr{C}$, $\mathcal{Z}_{1 \le i \le 3}$ are DBM
//$T_{ZG}$ is a set of transitions (i.e.  $\rightarrow_{ZG}= T_{ZG}$)
//D and Q are sets of pairs in $S \times \Phi^+(X)$
//D is the set of open states
DMZG BuildSymbZoneGraph(DMTA $\mathscr{C}$){
  $q_0$ = $(s_0, Extra^+_{LU}(\mathcal{Z}_0))$ s.t for all $x$ ∈ $X$ and $v$ ∈ $\mathcal{Z}_0$, $v(x) = 0$;
  Q, D = $\{q_0\}$, $T_{ZG}$ = $\emptyset$;
   while{D != $\emptyset$}{
    Choose and Remove $(s, \mathcal{Z}_1)$ from D;
     for(transition $e$ = $(s, a, \phi, Y, s')$ s.t $\mathcal{Z}_1 \wedge \phi \neq \emptyset$){
       //$\mathcal{Z}_2$ is the successor
       $\mathcal{Z}_2$ = $Extra^+_{LU}(\text{post}(\mathcal{Z}_1, e))$;
       $E_{ZG}$ = $E_{ZG}$ ∪ $\{e\}$;
       if($\exists(s', \mathcal{Z}_3) \in$ Q  $s.t$  $\mathcal{Z}_2 \subseteq \mathcal{Z}_3$){
         $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{e}_{ZG} (s', \mathcal{Z}_3)\}$;
       }
       else{
         $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{e}_{ZG} (s', \mathcal{Z}_2)\}$;
         Q = Q $\cup \{(s', \mathcal{Z}_2)\}$;
         D = D $\cup \{(s', \mathcal{Z}_2)\}$;
        }
     }
     $\mathcal{Z}_2$ = $Extra^+_{LU}((\mathcal{Z}_1 \uparrow_{R(s)} \wedge I(s)))$;
     if($\exists(s, \mathcal{Z}_3) \in$ Q  $s.t$  $\mathcal{Z}_2 \subseteq \mathcal{Z}_3$){
      $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{\epsilon}_{ZG} (s, \mathcal{Z}_3)\}$;
     }
     else{
      $T_{ZG}$ = $T_{ZG} \cup \{(s, \mathcal{Z}_1) \xrightarrow{\epsilon}_{ZG} (s', \mathcal{Z}_2)\}$;
      Q = Q $\cup \{(s, \mathcal{Z}_2)\}$;
      D = D $\cup \{(s, \mathcal{Z}_2)\}$;
     }
  }
return DMZG$(Q, q_0, (\Sigma \cup \{\epsilon\}), T_{ZG})$;
}
```

Algorithm C.1: Reachable (Derivative) Multi-timed Zone Graph with Subsumption.

```
Input: A DMZG($\mathscr{C}$)= $(Q, q_0 = (q^0_{\mathscr{A}}, q^0_{\mathscr{B}}), \Sigma = \Sigma_{\mathscr{A}} = \Sigma_{\mathscr{B}}, \rightarrow_{ZG})$, $\Pi_0 = Q$
Output: A stable set of zones $\Pi$
//$\Pi$ is a set of zones, $\mathcal{Z}$ are clock zones

ZG PartitionZoneGraph(ZG $\mathscr{C}$, $\Pi_0$){
  //Get the input set of zones $\Pi$
  $\Pi'$ = $\Pi_0$;
  do{
   // Refine $\Pi'$ by delay transitions
   for(each zone  $\mathcal{Z}$ $\in$ $\Pi'$){
     $\Pi'$ = TimeRefine($\mathcal{Z}$, $\Pi'$);
    }
   // Refine $\Pi'$ by discrete transitions
   for(each zone  $\mathcal{Z}$ $\in$ $\Pi'$) {
    $\Pi'$ $\leftarrow$ DiscreteSigRefine($\mathcal{Z}$, $\Pi'$);
   }
 }while($\Pi'$ does not change)
return $\Pi'$;
}
```

Algorithm C.2: The Refinement Algorithm for a Reachable DMZG.

TimePred$_{\uparrow_\psi}(\mathcal{Z}, \mathcal{Z}'_i)$, where $\mathcal{Z}'_i \in \Pi$, for $i = 1,2$. Since Π is a stable set of zones, $\mathcal{Z}$, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are all disjoint. Assumes $v \in \mathcal{Z}_1 \cap \mathcal{Z}_2$. For $i = 1,2$, $\exists\ \vec{d}_i \in \mathbb{R}^X_{>0}$ such that $v + \vec{d}_i \in \mathcal{Z}'_i$ and $\forall v \in \mathcal{Z}, \exists \vec{d}_i \in \mathbb{R}^X_{>0}, \exists\ \tau \in Rates, \exists\ t, t'' \geq 0$ and $t \leq t''$, $\forall t', t \leq t' \leq t''$, $(v + \vec{d}_i) \in \mathcal{Z}'_i$, $\vec{d}_i = \tau(t'') - \tau(t)$ and $\forall\ \vec{d}'_i, 0 \leq \vec{d}'_i \leq \vec{d}_i$ then $(v + \vec{d}'_i)$ $\in (\mathcal{Z} \cup \mathcal{Z}'_i)$, $\vec{d}'_i = \tau(t') - \tau(t)$. Observe that $\vec{d}_1 \neq \vec{d}_2$, since $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are disjoint. Without loss of generality, assume $\vec{d}_1 < \vec{d}_2$. We have that $v + \vec{d}_1 \in \mathcal{Z}'_1$ and $v + \vec{d}_1$ $\in \mathcal{Z} \cup \mathcal{Z}'_2$, that is, either $v + \vec{d}_1 \in \mathcal{Z}'_1 \cap \mathcal{Z}$ or $v + \vec{d}_1 \in \mathcal{Z}'_1 \cap \mathcal{Z}'_2$, which contradicts the fact that $\mathcal{Z}$, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are all disjoint. This proves that $\mathcal{Z}_1$ and $\mathcal{Z}_2$ are disjoint. Now, let $v \in \mathcal{Z}$. We can find $\mathbb{R}^X_{>0}$ and $\mathcal{Z}' \in \Pi$ such that $v + \vec{d} \in \mathcal{Z}'$ and $\forall v \in \mathcal{Z}, \exists \vec{d} \in \mathbb{R}^X_{>0}, \exists\ \tau \in Rates, \exists\ t, t'' \geq 0$ and $t \leq t''$, $\forall t', t \leq t' \leq t''$, $(v + \vec{d}) \in$ $\mathcal{Z}'$, $\vec{d} = \tau(t'') - \tau(t)$ and $\forall\ \vec{d}', 0 \leq \vec{d}' \leq \vec{d}$ then $(v + \vec{d}') \in (\mathcal{Z} \cup \mathcal{Z}')$, $\vec{d}' = \tau(t') - \tau(t)$. By definition, $v \in$ TimePred$_{\uparrow_\psi}(\mathcal{Z}, \mathcal{Z}')$.

(ii) Now, consider Π_2 = DiscreteSigRefine$(\mathcal{Z}, \Pi)$. For all members of Π_2 are zones. By the distributivity of pred over union (pred$(\mathcal{Z}_1 \cup \mathcal{Z}_2, e) \downarrow_\psi$ = pred$(\mathcal{Z}_1, e) \downarrow_\psi$ $\cup$ pred$(\mathcal{Z}_2, e)) \downarrow_\psi$ is a zone of $\mathcal{Z}$. It remains to show that they are disjoint. Let $\mathcal{Z}_i \in \Pi_2$, $\mathcal{Z}_i = \mathcal{Z} \cap$ pred$(\mathcal{Z}'_i, e) \downarrow_\psi$, where $\mathcal{Z}'_i \in \Pi$, for $i = 1,2$. Since Π is a stable set of zones, $\mathcal{Z}'_1$ and $\mathcal{Z}'_2$ are disjoint. Assume $v \in \mathcal{Z}_1 \cap \mathcal{Z}_2$. Recall that the successor of v, say v', is unique. Since $v \in$ pred$(\mathcal{Z}'_1, e) \downarrow_\psi \cap$ pred$(\mathcal{Z}'_2, e) \downarrow_\psi$, it must be that $v' \in \mathcal{Z}'_1 \cap \mathcal{Z}'_2$, which contradicts $\mathcal{Z}'_1 \cap \mathcal{Z}'_2 = \emptyset$.

□

APPENDIX D

TOOLS USER MANUALS

D.1 MULTI-TEMPO Tool

First, we introduce the syntax using a simple example for readers familiar with MULTI-TEMPO, and not interested in subtle details (such as the synchronization model). A formal definition of MULTI-TEMPO can be found in Section 8.6.3.

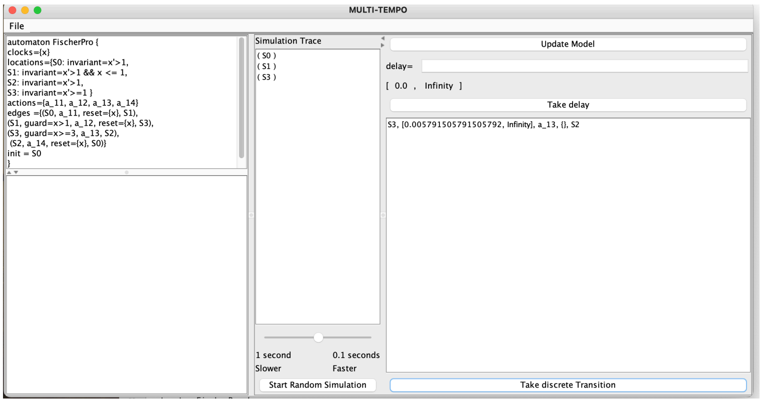
D.1.1 Fischer's Protocol in MULTI-TEMPO

We use Fischer's protocol presented in Section 8.5.4 as a motivating example. This version of the protocol is neither the most complete, nor the most simple. We just use it here to introduce various aspects of the MULTI-TEMPO input syntax. We give below this model using the MULTI-TEMPO syntax. This model is given in Figure 8.11.

```
automaton FischerPro {
clocks={x}
locations={S0: invariant=x’=1,
S1: invariant=x’>1 && x <= 1,
S2: invariant=x’> 1,
S3: invariant=x’=1}
actions={a_11, a_12, a_13, a_14}
edges ={(S0, a_11, reset={x}, S1),
(S1, guard=x>1, a_12, reset={x}, S3),
(S3, guard=x>=3, a_13, S2),
(S2, a_14, reset={x}, S0)}
init = S0
}
```

Figure D.1: Fischer's protocol in MULTI-TEMPO

- Clock declarations: The clock declarations start with the keyword clock. This model contains the clock: x.
- Location declarations: The location declarations start with the keyword locations. This model contains the locations: S0, S1, S2, S3. The automaton is initially in location S0, with rate invariant $x' = 1$ and no invariant (depicted by invariant true). In location S1 the rate invariant is $x' > 1$ and $x <= 1$. In location S3 the rate invariant is $x' = 1$ and no invariant (depicted by invariant true). In location S2 the rate invariant is $x' > 1$ and no invariant (depicted by invariant true).
- Action declarations: The action declarations start with the keyword actions. This model contains the actions: a_11, a_12, a_13, a_14.
- Edge declarations: The edge declarations start with the keyword edges. The edges are formed by the source and destination locations, the guards, the action, and the clock reset (e.g., $(S1, guard = x > 1, a\_12, reset = x, S3)$).
- Init declaration: The initial location star with the keyword init. This model contains the initial location: S0.



```
automaton FischerPro {
clocks={x}
locations={S0: invariant=x'>1,
S1: invariant=x'>1 && x <= 1,
S2: invariant=x'>1,
S3: invariant=x'>=1 }
actions={a_11, a_12, a_13, a_14}
edges ={(S0, a_11, reset={x}, S1),
(S1, guard=x>1, a_12, reset={x}, S3),
(S3, guard=x>=3, a_13, S2),
 (S2, a_14, reset={x}, S0)}
init = S0
}
```

```
( S0 )
( S1 )
( S3 )
```

```
[  0.0  ,   Infinity  ]
```

```
S3, [0.005791505791505792, Infinity], a_13, {}, S2
```

Figure D.2: Graphical view MULTI-TEMPO

D.2 MUTES Tool

To run our MUTES tool, it is required that you have two DMTA created with UPPAAL (DMTA $\mathscr{A}$ and DMTA $\mathscr{B}$). A formal definition of DMTA can be found in Section 8.1 (see Figure D.3).

Now, we can run MUTES with two DMTA. Assuming these two models (input files automaton1.xml and automaton2.xml), the command calling MUTES is as follows:

```
$ java -cp MUTEs.jar  automaton1.xml automaton2.xml
```

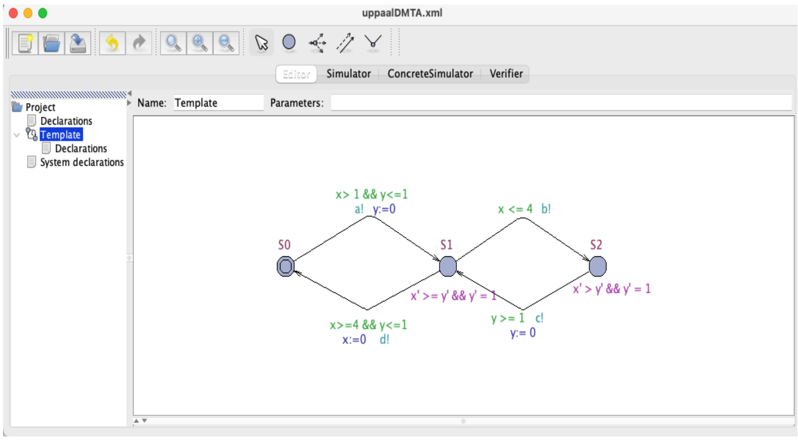
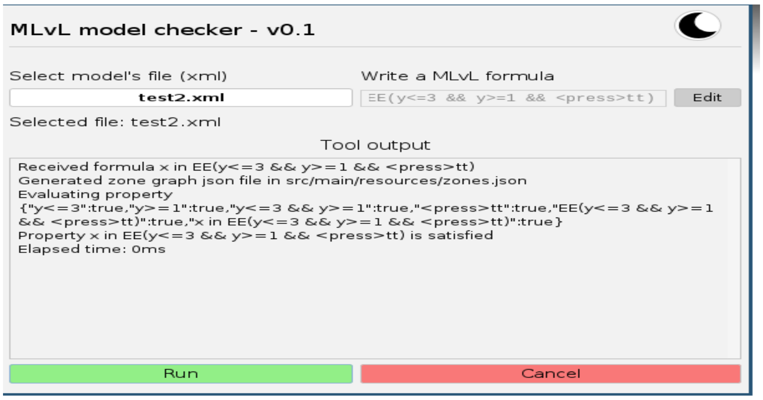


Figure D.3: DMTA Example

D.3 MIMETIC Tool

To run our MIMETIC tool, it is required that you have a DMTA created with UPPAAL (DMTA $\mathscr{A}$). A formal definition of DMTA can be found in Section 8.1. After starting the tool, the first thing to do is to choose the automaton. This is done via the respective load buttons, as depicted in Figure D.4


MLvL model checker - v0.1

Select model's file (xml)

test2.xml

Write a MLvL formula

EE(y<=3 && y>=1 && <press>tt)

Selected file: test2.xml

Tool output

```
Received formula x in EE(y<=3 && y>=1 && <press>tt)
Generated zone graph json file in src/main/resources/zones.json
Evaluating property
{"y<=3":true,"y>=1":true,"y<=3 && y>=1":true,"<press>tt":true,"EE(y<=3 && y>=1
&& <press>tt)":true,"x in EE(y<=3 && y>=1 && <press>tt)":true}
Property x in EE(y<=3 && y>=1 && <press>tt) is satisfied
Elapsed time: 0ms
```

Figure D.4: MIMETIC Tool (Property is satified)

After clicking the button, a dialogue will pop up. Here, you can select the XML file of the DMTA. The chosen file is displayed in the window below the text field. The formula can be writing in the text field next to the Edit button (e.g., $y\ in\ EE(y <= 3\ \&\&\ y >= 1\ \&\&\ < a > tt)\ \&\&\ (x >= 1\ \&\& < a > tt)$, see grammar in Section 8.6.4).

Appendix E

Acronyms

APTL Alternative Propositional Temporal Logic

BA Büchi Automata

CAN Controller Area Network

ClockTL Clock Temporal Logic

CED Complex Event Detection

CTL Computation Tree Logic

DBA Deterministic Büchi Automata

DCD Distributed Clock Derivatives

DEC Distributed Event Clock

DECA Distributed (Recursive) Event Clock Automata

DECTL Distributed Recursive Event Clock Temporal Logic

DBM Difference Bound Matrix

DDBM Derivative Difference Bound Matrix

DFA Deterministic Finite Automata

DML$_v$ Multi-timed Hennessy-Milner Logic with Clock Derivatives

DMTA Timed Automata with Clock Derivatives

DS Distributed Systems

DRTL Distributed Real-Time Temporal Logic

DRTS Distributed Real-Time System

DTA Distributed Timed Automata

DTM Deterministic Turing Machine

EC Event Clock

ECA Event Clock Automata

ETL Extension Temporal Logic

EventClockTL Recursive Event Clock Temporal Logic

FOML First Order Monadic Logic

FM Formal Methods

FA Finite Automata

HML Hennessy-Milner Logic

HA Hybrid Automata

icTA Timed Automata with Independent Clocks

KRONOS Model Checking Tool

LBA Labeled Büchi Automata

Lv Timed Modal Logic

LTL Linear Temporal Logic

LTS Labeled Transition System

MIC Multiple Independent Clocks

MIMETIC Multi-timed Model Checking Tool

MITL Metric Interval Temporal Logic

ML$_v$ Multi-timed Hennessy-Milner Logic

MTA Multi-timed Automata

MTL Metric Temporal Logic

MITL Metric Interval Temporal Logic

MULTI-TEMPO Multi-timed Automata Tool

MUTES Multi-timed Bisimulation Tool

MZG Mult-timed Zone Graph

NBA Nondeterministic Büchi Automata

NFA Nondeterministic Finite Automata

NTM Nondeterministic Turing Machine

RG Region Automaton

RECA Recursive Event Clock Automata

RTS Real-Time Systems

SOL Second-Order Logic

TA Timed Automata

TEMPO Model Checking Tool

TT Timed Trace

TIS Timed Interval Sequence

TCTL Timed Computation Tree Logic

TLC Temporal Logic with Counting

TLTL Timed Linear Temporal Logic

TLTS Timed Labelled Transition Systems

TPN Timed Petri Nets

TPTL Timed Propositional Temporal Logic

TTS Timed Transition System

TM Turing Machine

UPPAAL Model Checking Tool

WirelessHART Wireless Network

BIBLIOGRAPHY

[ABG$^+$08] S. Akshay, Benedikt Bollig, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Distributed timed automata with independently evolving clocks. In **19th International Conference, CONCUR 2008**, volume 5201 of **Lecture Notes in Computer Science**, pages 82–97. Springer, 2008.

[ABS01] Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. Trex: A tool for reachability analysis of complex systems. In **CAV**, volume 2102 of **Lecture Notes in Computer Science**, pages 368–372. Springer, 2001.

[ACD93] Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. **Information and Computation.**, 104(1):2–34, 1993.

[ACH$^+$95] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. **Theoretical Computer Science**, 138(1):3–34, 1995.

[AD94] Rajeev Alur and David L. Dill. A theory of timed automata. **Theoretical Computer Science**, 126(2):183–235, 1994.

[AFH94] Rajeev Alur, Limor Fix, and Thomas A. Henzinger. A determinizable class of timed automata. In **CAV**, volume 818 of **Lecture Notes in Computer Science**, pages 1–13. Springer, 1994.

[AFH96] Rajeev Alur, Tomás Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. **J. ACM**, 43(1):116–146, 1996.

[AGGS22] S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan. Simulations for Event-Clock Automata. In Bartek Klin, Sławomir Lasota, and Anca Muscholl, editors, **In 33rd International Conference on Concurrency Theory, CONCUR 2022)**, volume 243 of **Leibniz International Proceedings in Informatics (LIPIcs)**, pages 13:1–13:18, Dagstuhl, Germany, 2022.

[AH91] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In **REX Workshop**, pages 74–106, 1991.

[AH93] Rajeev Alur and Thomas A. Henzinger. Real-time logics: Complexity and expressiveness. **Information and Computation**, 104(1):35–77, May 1993.

[AHM$^+$98] Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram Rajamani, and Serdar Taşiran. MOCHA: Modularity in model checking. In Alan J. Hu and Moshe Y. Vardi, editors, **In Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)**, volume 1427 of **Lecture Notes in Computer Science**, pages 521–525. Springer-Verlag, June 1998.

[AKM03] Yasmina Abdeddaïm, Abdelkarim Kerbaa, and Oded Maler. Task graph scheduling using timed automata. In **Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03)**, page 237. IEEE Comp. Soc. Press, April 2003.

[AL99] Luca Aceto and François Laroussinie. Is your model checker on time? on the complexity of model checking for timed modal logics. In **MFCS**, pages 125–136. Springer, 1999.

[Alu92] Rajeev Alur. **Techniques for automatic verification of real-time systems**. PhD thesis, Stanford University, Stanford, CA, USA, 1992.

[Ash03] Neil Ashby. Relativity in the global positioning system. **In Living Reviews in Relativity**, 2003.

[Asp18] Mikael Asplund. Automatically proving the correctness of vehicle coordination. **ICT Express**, 4(1):51–54, 2018.

[AT05] Karine Altisen and Stavros Tripakis. Implementation of timed automata: An issue of semantics or modeling? In **FORMATS**, pages 273–288, 2005.

[BAMP81] Mordechai Ben-Ari, Zohar Manna, and Amir Pnueli. The temporal logic of branching time. In **POPL**, pages 164–176, 1981.

[BBFL03] Gerd Behrmann, Patricia Bouyer, Emmanuel Fleury, and Kim Guldstrand Larsen. Static guard analysis in timed automata verification. In Hubert Garavel and John Hatcliff, editors, **TACAS**, volume 2619 of **Lecture Notes in Computer Science**, pages 254–277. Springer, 2003.

[BBL$^+$22] Davide Basile, Maurice H. ter Beek, Sami Lazreg, Maxime Cordy, and Axel Legay. Static detection of equivalent mutants in real-time model-based mutation testing. **Empirical Software Engineering**, 27(7):160, 2022.

[BBLP04] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone based abstractions of timed automata. In Kurt Jensen and Andreas Podelski, editors, **Tools and**

Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings, volume 2988 of **Lecture Notes in Computer Science**, pages 312–326. Springer, 2004.

[BBLP06] Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. **Int. J. Softw. Tools Technol. Transf.**, 8(3):204–215, 2006.

[BC13] Sandie Balaguer and Thomas Chatain. Avoiding shared clocks in networks of timed automata. In **International Conference on Concurrency Theory**, volume 7454 of **Lecture Notes in Computer Science**, pages 100–114, 2013.

[BCDL09] Peter E. Bulychev, Thomas Chatain, Alexandre David, and Kim Guldstrand Larsen. Efficient on-the-fly algorithm for checking alternating timed simulation. In **FORMATS**, volume 5813 of **Lecture Notes in Computer Science**, pages 73–87. Springer, 2009.

[BCG88] Michael C. Browne, Edmund M. Clarke, and Orna Grumberg. Characterizing finite kripke structures in propositional temporal logic. **Theoretical Computer Science**, 59:115–131, 1988.

[BDL$^+$06] Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In **Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems**, QEST '06, 2006.

[BDM$^+$98] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In **CAV**, volume 1427 of **Lecture Notes in Computer Science**, pages 546–550. Springer, 1998.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema. **Modal Logic**, volume 53 of **Cambridge Tracts in Theoretical Computer Science**. Cambridge University Press, 2001.

[BEG$^+$16] Thomas Brihaye, Morgane Estiévenart, Gilles Geeraerts, Hsi-Ming Ho, Benjamin Monmege, and Nathalie Sznajder. Real-time synthesis is hard! **CoRR**, abs/1606.07124, 2016.

[Ber02] Martin Berger. **Towards Abstractions for Distributed Systems**. PhD thesis, Imperial College, Dept. of Computing, 2002.

[Ber04] Martin Berger. Basic theory of reduction congruence for two timed asynchronous π-calculi. In **In 15th International Conference on Concurrency Theory, CONCUR 2004**, 2004.

[BFLM11] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Quantitative analysis of real-time systems using priced timed automata. **Communications of the ACM**, 54(9):78–87, September 2011.

[BG85] Timothy Alan Budd and Ajei Sarat Gopal. Program testing by specification mutation. **Computer Languages**, 10(1):63–73, 1985.

[BGH+22] Patricia Bouyer, Paul Gastin, Frédéric Herbreteau, Ocan Sankur, and B. Srivathsan. Zone-based verification of timed automata: Extrapolations, simulations and what next? In Sergiy Bogomolov and David Parker, editors, **Formal Modeling and Analysis of Timed Systems - 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings**, volume 13465 of **Lecture Notes in Computer Science**, pages 16–42. Springer, 2022.

[BGHM17a] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. Mightyl: A compositional translation from MITL to timed automata. In Rupak Majumdar and Viktor Kuncak, editors, **Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I**, volume 10426 of **Lecture Notes in Computer Science**, pages 421–440. Springer, 2017.

[BGHM17b] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. Timed-automata-based verification of MITL over signals. In **TIME**, volume 90 of **LIPIcs**, pages 7:1–7:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BGK+02] Johan Bengtsson, W. O. David Griffioen, Kåre J. Kristoffersen, Kim Guldstrand Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. Automated verification of an audio-control protocol using UPPAAL. **J. Log. Algebraic Methods Program.**, 52-53:163–181, 2002.

[BGO+04] Marius Bozga, Susanne Graf, Ileana Ober, Iulian Ober, and Joseph Sifakis. The IF toolset. In **SFM**, volume 3185 of **Lecture Notes in Computer Science**, pages 237–267. Springer, 2004.

[BJLY98] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In **In 9th International Conference on Concurrency Theory, CONCUR 1998**, pages 485–500, 1998.

[BK08] Christel Baier and Joost-Pieter Katoen. **Principles of Model Checking (Representation and Mind Series)**. The MIT Press, 2008.

[BLR05] Patricia Bouyer, François Laroussinie, and Pierre-Alain Reynier. Diagonal constraints in timed automata: Forward analysis of timed systems. In **In 3rd International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2005**, pages 112–126, Berlin, Heidelberg, 2005. Springer-Verlag.

[BLT94] Tommaso Bolognesi, Ferdinando Lucidi, and Sebastiano Trigila. Converging towards a timed lotos standard. **In Computer Standard Interfaces**, pages 87–118, 1994.

[BO02] Stefan Blom and Simona Orzan. A distributed algorithm for strong bisimulation reduction of state spaces. **Electronic Notes in Theoretical Computer Science**, 68(4):523–538, 2002.

[Bou04a] Patricia Bouyer. Forward analysis of updatable timed automata. **Formal Methods in System Design**, 24(3):281–320, 2004.

[Bou04b] Patricia Bouyer. Forward analysis of updatable timed automata. **Formal Methods Syst. Des.**, 24(3):281–320, 2004.

[BPDG98] Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. **Fundam. Inf.**, 36(2-3):145–182, November 1998.

[BRS13] Marcello M. Bersani, Matteo Rossi, and Pierluigi San Pietro. A tool for deciding the satisfiability of continuous-time metric temporal logic. In César Sánchez, Kristen Brent Venable, and Esteban Zimányi, editors, **2013 20th International Symposium on Temporal Representation and Reasoning, Pensacola, FL, USA, September 26-28, 2013**, pages 99–106. IEEE Computer Society, 2013.

[BtBCL20] Davide Basile, Maurice H. ter Beek, Maxime Cordy, and Axel Legay. Tackling the equivalent mutant problem in real-time systems: the 12 commandments of model-based mutation testing. In Roberto Erick Lopez-Herrejon, editor, **SPLC ’20: 24th ACM International Systems and Software Product Line Conference, Montreal, Quebec, Canada, October 19-23, 2020, Volume A**, pages 30:1–30:11. ACM, 2020.

[Büc62] Julius R. Büchi. On a decision method in restricted second order arithmetic. In **LMPS’60**, pages 1–11, 1962.

[BY03] Johan Bengtsson and Wang Yi. On clock difference constraints and termination in reachability analysis of timed automata. In **ICFEM**, volume 2885 of **Lecture Notes in Computer Science**, pages 491–503. Springer, 2003.

[BY04] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In **Lecture Notes on Concurrency and Petri Nets**, 2004.

[CAS01] Aurore Collomb-Annichini and Mihaela Sighireanu. Parameterized Reachability Analysis of the IEEE 1394 Root Contention Protocol using TReX. In Paul Pattersson and Sergio Yovine, editors, **Proceedings of the Real-Time Tools Workshop (RT TOOLS’01)**, TR 2001-014, pages 1–20, Aalborg, Denmark, August 2001.

[CE82] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In **Logic of Programs, Workshop**, pages 52–71, 1982.

[Cer93] Karlis Cerans. Decidability of bisimulation equivalences for parallel timer processes. In **Proceedings of the Fourth International Workshop on Computer Aided Verification**, pages 302–315, London, UK, 1993.

[CH97] Cristian Calude and Juraj Hromkovič. **Complexity: a language-theoretic point of view**, pages 1–60. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[CKS81] Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. **In ACM**, pages 114–133, 1981.

[CL00] Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In Catuscia Palamidessi, editor, **CONCUR 2000 - Concurrency Theory, 11th International Conference, University Park, PA, USA, August 22-25, 2000, Proceedings**, volume 1877 of **Lecture Notes in Computer Science**, pages 138–152. Springer, 2000.

[Cri96] Flaviu Cristian. Synchronous and asynchronous. **Commun. ACM**, 39(4):88–97, April 1996.

[D'A99] Pedro Ruben D'Argenio. **Algebras and Automata for Timed and Stochastic Systems**. PhD thesis, Imperial College, Dept. of Computing, Enschede, November 1999.

[dBdRR89] J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors. **In Rex Workshop**, volume 354, 1989.

[DCBB19] Mahieddine Dellabani, Jacques Combaz, Saddek Bensalem, and Marius Bozga. Local planning semantics: A semantics for distributed real-time systems. **Leibniz Trans. Embed. Syst.**, 6:01:1–01:27, 2019.

[DGLM99] Dennis Dams, Rob Gerth, Stefan Leue, and Mieke Massink, editors. **in SPIN'99**, volume 1680, 1999.

[Dil90] David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In Joseph Sifakis, editor, **Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems (AVMFSS'89)**, volume 407 of **Lecture Notes in Computer Science**, pages 197–212. Springer-Verlag, 1990.

[Dim03] Catalin Dima. Distributed real-time automata. In C. Martin-Vide and V. Mitrana, editors, **Essays in honor of Gheorghe Păun**, pages 131–140. Springer, 2003.

[Dim07] Catalin Dima. Dynamical properties of timed automata revisited. In **FORMATS**, volume 4763 of **Lecture Notes in Computer Science**, pages 130–146. Springer, 2007.

[DK06] Conrado Daws and Piotr Kordy. Symbolic robustness analysis of timed automata. In **FORMATS**, volume 4202 of **Lecture Notes in Computer Science**, pages 143–155. Springer, 2006.

[DL07] Catalin Dima and Ruggero Lanotte. Distributed time-asynchronous automata. In **ICTAC**, pages 185–200, 2007.

[DLL$^+$10] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman, and Andrzej Wasowski. Timed i/o automata: A complete specification theory for real-time systems. In **Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control**, HSCC '10, New York, NY, USA, 2010. ACM.

[DLL$^+$11] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikucionis, and Zheng Wang. Time for statistical model checking of real-time systems. In **CAV**, volume 6806 of **Lecture Notes in Computer Science**, pages 349–355. Springer, 2011.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In T.A. Henzinger and E.D. Sontag, editors, **Hybrid Systems III, Verification and Control**, pages 208–219. Lecture Notes in Computer Science 1066, Springer Verlag, 1996.

[Doy06] L. Doyen. **Algorithmic Analysis of Complex Semantics for Timed and Hybrid Automata**. PhD thesis, Université Libre de Bruxelles, 2006.

[DT98] Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In Bernhard Steffen, editor, **Tools and Algorithms for Construction and Analysis of Systems, 4th International Conference, TACAS '98, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings**, volume 1384 of **Lecture Notes in Computer Science**, pages 313–329. Springer, 1998.

[DW07] Martin De Wulf. **From Timed Models to Timed Implementations**. PhD thesis, Université Libre de Bruxelles, Belgium, 2007.

[DWDMR04] Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robustness and implementability of timed automata. In **FORMATS**, pages 118–133, 2004.

[DY96] Conrado Daws and Sergio Yovine. Reducing the number of clock variables of timed automata. In **RTSS**, pages 73–81. IEEE Computer Society, 1996.

[EH86] E. Allen Emerson and Joseph Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. **J. ACM**, 33(1):151–178, 1986.

[Eme90] E. Allen Emerson. Handbook of theoretical computer science (vol. b). In Jan van Leeuwen, editor, **CONCUR**, chapter Temporal and modal logic, pages 995–1072. MIT Press, Cambridge, MA, USA, 1990.

[ERGM09] Nancy El Rachkidy, Alexandre Guitton, and Michel Misson. Optimizing the setup phase of an ieee 802.15.4 wireless sensor network. In **IFIP conference on Wireless days**, WD'09, pages 57–61, 2009.

[FA14] Gordon Fraser and Andrea Arcuri. Achieving scalable mutation-based generation of whole test suites. **Empirical Software Engineering**, pages 1–30, 2014.

[FL79] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. **In Comput. Syst. Sci.**, pages 194–211, 1979.

[GMP04] Marí;a del Mar Gallardo, Pedro Merino, and Ernesto Pimentel. A generalized semantics of promela for abstract model checking. **In Form. Asp. Comput.**, pages 166–193, 2004.

[GMP13] Chryssis Georgiou, Peter M. Musial, and Christos Ploutarchou. Tempo-toolkit: Tempo to java translation module. In **2013 IEEE 12th International Symposium on Network Computing and Applications, Cambridge, MA, USA, August 22-24, 2013**. IEEE Computer Society, 2013.

[GMS18] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. **CoRR**, abs/1806.11007, 2018.

[GMS19] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In Isil Dillig and Serdar Tasiran, editors, **Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I**, volume 11561 of **Lecture Notes in Computer Science**, pages 41–59. Springer, 2019.

[GMS20] Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability for updatable timed automata made faster and more effective. In Nitin Saxena and Sunil Simon, editors, **40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)**, volume 182 of **LIPIcs**, pages 47:1–47:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[GPSS80] Dov Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal analysis of fairness. In **POPL '80**, pages 163–173, 1980.

[GRS11] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Event-clock automata: From theory to practice. **CoRR**, abs/1107.4138, 2011.

[GRS14] Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. On regions and zones for event-clock automata. **Formal Methods in System Design**, 45(3):330–380, December 2014.

[GYH$^+$22] Qi Guo, Wangyang Yu, Fei Hao, Yuke Zhou, and Yuan Liu. Modelling and analysis of adaptive cruise control system based on synchronization theory of petri nets. **Electronics**, 11(21), 2022.

[Hen91] Thomas A. Henzinger. **The Temporal Specification and Verification of Real-Time Systems**. PhD thesis, Stanford University, Palo Alto, California, USA, 1991.

[Hen96] Thomas A. Henzinger. The theory of hybrid automata. In **Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996**, pages 278–292. IEEE Computer Society, 1996.

[HHK95] Monika R. Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In **Proceedings of the 36th Annual Symposium on Foundations of Computer Science**, FOCS '95, Washington, DC, USA, 1995. IEEE Computer Society.

[HHWT97] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. Hytech: A model checker for hybrid systems. **Int. J. Softw. Tools Technol. Transf.**, 1, 1997.

[Hie04] Rob. Hierons. Testing from a nondeterministic finite state machine using adaptive state counting. **In IEEE Trans. Comput.**, pages 1330–1342, 2004.

[HKPV98] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? **J. Comput. Syst. Sci.**, 57(1):94–124, 1998.

[HL94] Constance L. Heitmeyer and Nancy A. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In **RTSS**, pages 120–131. IEEE Computer Society, 1994.

[HM85] Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. **In ACM**, pages 137–161, 1985.

[HMP91] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In **REX Workshop**, pages 226–251, 1991.

[HMP92] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. What good are digital clocks? In **ICALP'92**, pages 545–558, New York, NY, USA, 1992. ACM.

[HMU06] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. **Introduction to Automata Theory, Languages, and Computation (3rd Edition)**. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[HNSY94] Thomas A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. **Inf. Comput.**, 111(2):193–244, June 1994.

[How76] William E. Howden. Reliability of the path analysis testing strategy. **IEEE Transactions on Software Engineering**, 2(3):208–215, 1976.

[HPV00] Klaus Havelund, John Penix, and Willem Visser, editors. **SPIN Model Checking and Software Verification**, volume 1885, 2000.

[HR95] Matthew Hennessy and Tim Regan. A process algebra for timed systems. **In Inf. Comput.**, pages 221–239, 1995.

[HR04] Yoram Hirshfeld and Alexander Moshe Rabinovich. Logics for real time: Decidability and complexity. **Fundam. Informaticae**, 62(1):1–28, 2004.

[HR06] Yoram Hirshfeld and Alexander Moshe Rabinovich. An expressive temporal logic for real time. In **MFCS**, volume 4162 of **Lecture Notes in Computer Science**, pages 492–504. Springer, 2006.

[HRS98] Thomas A. Henzinger, Jean-François Raskin, and Pierre-Yves Schobbens. The regular real-time languages. In **ICALP**, pages 580–591, 1998.

[HW11] Douglas Hanahan and Robert A. Weinberg. Hallmarks of cancer: The next generation. **Cell**, 144(5):646–674, 2011.

[HW16] Douglas Hanahan and Robert A. Weinberg. The hallmarks of cancer: Perspectives for cancer medicine. In **Oxford Textbook of Oncology**. Oxford University Press, 01 2016.

[JH11] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. **IEEE Trans. Softw. Eng.**, 37(5):649–678, September 2011.

[JLMX14] Samy Jaziri, Kim Guldstrand Larsen, Radu Mardare, and Bingtian Xue. Adequacy and complete axiomatization for timed modal logic. **Electr. Notes Theor. Comput. Sci.**, 308:183–210, 2014.

[JLS96] Henrik Ejersbo Jensen, Kim G. Larsen, and Arne Skou. Modelling and analysis of a collision avoidance protocol using spin and uppaal. In Jean-Charles Grégoire, Gerard J. Holzmann, and Doron A. Peled, editors, **The Spin Verification System, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, August, 1996**, volume 32 of **DIMACS Series in Discrete Mathematics and Theoretical Computer Science**, pages 33–49. DIMACS/AMS, 1996.

[JLS08] Marcin Jurdzinski, François Laroussinie, and Jeremy Sproston. Model checking probabilistic timed automata with one or two clocks. **CoRR**, abs/0809.0060, 2008.

[KAH06] Hermann Kopetz, Astrit Ademaj, and Alexander Hanzlik. Combination of clock-state and clock-rate correction in fault-tolerant distributed systems. **In Real-Time Syst.**, pages 139–173, 2006.

[KLSV03] Dilsun K. Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. Timed i/o automata: A mathematical framework for modeling and analyzing real-time systems. In **Proceedings of the 24th IEEE International Real-Time Systems Symposium**, RTSS '03, pages 166–, Washington, DC, USA, 2003. IEEE Computer Society.

[KLSV10] Dilsun Kaynar, Nancy Lynch, Roberto Segala, and Frits Vaandrager. **The Theory of Timed I/O Automata, Second Edition**. Morgan & Claypool Publishers, 2nd edition, 2010.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. **In Real-Time Systems**, pages 255–299, 1990.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. **Theor. Comput. Sci.**, 27:333–354, 1983.

[Kri99] Padmanabhan Krishnan. Distributed timed automata. **In Electr. Notes Theor. Comput. Sci.**, pages 185–200, 1999.

[LAS] The Liége Automata-based Symbolic Handler (LASH). Available at http://www.montefiore.ulg.ac.be/~boigelot/research/lash/.

[LDPM20] Thierry Lecomte, David Deharbe, Etienne Prun, and Erwan Mottin. Applying a formal method in industry: a 25-year trajectory, 2020.

[LLNN17] K. G. Larsen, F. Lorber, B. Nielsen, and U. M. Nyman. Mutation-based test-case generation with ecdar. In **2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**, pages 319–328, March 2017.

[LLW95] François Laroussinie, Kim Guldstrand Larsen, and Carsten Weise. From timed automata to logic - and back. In **MFCS'95**, pages 529–539, 1995.

[LM14a] Kim G. Larsen and Radu Mardare. Complete proof systems for weighted modal logic. **Theor. Comput. Sci.**, 546:164–175, August 2014.

[LM14b] Kim G. Larsen and Radu Mardare. Complete proof systems for weighted modal logic. **Theor. Comput. Sci.**, 546:164–175, 2014.

[LPY98] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gear Controller. In **Proceedings of the Fourth International Conference on Tools and Algorithms for the Construction and Analysis of Systems**, volume 1384 of **Lecture Notes in Computer Science**, pages 281–297. Springer-Verlag, 1998.

[LS00] François Laroussinie and Philippe Schnoebelen. The state-explosion problem from trace to bisimulation equivalence. In Jerzy Tiuryn, editor, **Proceedings of the 3rd International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2000)**, volume 1784 of **Lecture Notes in Computer Science**, Berlin, Germany, 2000. Springer.

[LSV03] Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid i/o automata. **Inf. Comput.**, pages 105–157, 2003.

[Mah05] P. Mahata. **Model Checking Parameterized Timed Systems**. PhD thesis, Dept. of Information Technology, Uppsala University, Sweden, Uppsala, Sweden, 2005.

[Mer74] P.M. Merlin. **A Study of the Recoverability of Computing Systems**. University of California, Irvine, 1974.

[Mil89] Robin Milner. **Communication and Concurrency**. Prentice Hall, 1989. ISBN 0-13-114984-9 (Hard) 0-13-115007-3 (Pbk).

[MNP06] Oded Maler, Dejan Nickovic, and Amir Pnueli. From mitl to timed automata. In **FORMATS**, volume 4202 of **LNCS**, pages 274–289. Springer, 2006.

[MP90] R Mall and LM Patnaik. Specification and verification of timing properties of distributed real-time systems, 1990.

[MP92] Zohar Manna and Amir Pnueli. **The temporal logic of reactive and concurrent systems**. Springer-Verlag New York, Inc., New York, NY, USA, 1992.

[MP95] Zohar Manna and Amir Pnueli. **Temporal verification of reactive systems: safety**. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[MVS+18] Curtis Madsen, Prashant Vaidyanathan, Sadra Sadraddini, Cristian Ioan Vasile, Nicholas A. DeLateur, Ron Weiss, Douglas Densmore, and Calin Belta. Metrics for signal temporal logic formulae. **CoRR**, abs/1808.03315, 2018.

[MW84] Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. **In ACM**, pages 68–93, 1984.

[OAS17] James Jerson Ortiz, Moussa Amrani, and Pierre-Yves Schobbens. Multi-timed bisimulation for distributed timed automata. In Clark Barrett, Misty Davies, and Temesghen Kahsai, editors, **NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings**, volume 10227 of **Lecture Notes in Computer Science**, 2017.

[Off11] Jeff Offutt. A mutation carol: Past, present and future. **Information and Software Technology**, 53(10):1098–1107, October 2011.

[OLS10] James Ortiz, Axel Legay, and Pierre-Yves Schobbens. Memory event clocks. In **FORMATS'10**, pages 198–212, 2010.

[OLS11] James Jerson Ortiz, Axel Legay, and Pierre-Yves Schobbens. Distributed event clock automata - extended abstract. In **CIAA'11**, pages 250–263, 2011.

[OS06] Jo$\ddot{e}$l Ouaknine and Steve Schneider. Timed CSP: A retrospective. **In Electr. Notes Theor. Comput. Sci.**, pages 273–276, 2006.

[OW07] Joël Ouaknine and James Worrell. On the decidability and complexity of metric temporal logic over finite words. **In CoRR**, 2007.

[OWM12] Loes Olde Loohuis, Andreas Witzel, and Bud Mishra. Towards Cancer Hybrid Automata. **arXiv e-prints**, page arXiv:1208.3857, August 2012.

[PAM+22] Michele Persico, Claudia Abbruzzese, Silvia Matteoni, Paola Matarrese, Anna Maria Campana, Veronica Villani, Andrea Pace, and Marco G. Paggi. Tackling the behavior of cancer cells: Molecular bases for repurposing antipsychotic drugs in the treatment of glioblastoma. **Cells**, 11(2), 2022.

[Pap94] Christos M. Papadimitriou. **Computational complexity**. Addison-Wesley, Reading, Massachusetts, 1994.

[Par84] Rohit Parikh. Logics of knowledge, games and dynamic logic. In **FSTTCS**, pages 202–222, 1984.

[Par13] Terence Parr. **The Definitive ANTLR 4 Reference**. Pragmatic Bookshelf, Raleigh, NC, 2 edition, 2013.

[Pet81] James Lyle Peterson. **Petri Net Theory and the Modeling of Systems**. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1981.

[Pit06] Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In **Symposium on Logic in Computer Science**, pages 255–264, 2006.

[PJHL15] Mike Papadakis, Yue Jia, Mark Harman, and Yves Le Traon. Trivial compiler equivalence: A large scale empirical study of a simple fast and effective equivalent mutant detection technique. In **International Conference on Software Engineering, ICSE**, pages 936–946. IEEE, 2015.

[PKZ$^+$18] Mike Papadakis, Marinos Kintis, Jie Zhang, Yue Jia, Yves Le Traon, and Mark Harman. Mutation testing advances: An analysis and survey. **Advances in Computers**, 112, 2018.

[PM10] Mike Papadakis and Nicos Malevris. Automatic mutation test case generation via dynamic symbolic execution. In **ISSRE**, pages 121–130. IEEE Computer Society, 2010.

[Pnu77] Amir Pnueli. The temporal logic of programs. In **FOCS**, pages 46–57, 1977.

[PSR94] Boaz Patt-Shamir and Sergio Rajsbaum. A theory of clock synchronization. In **Proceedings of the 26th Annual ACM Symposium on Theory of Computing**, pages 810–819. ACM, 1994.

[PT87] Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. **SIAM Journal on Computing**, 16(6):973–989, 1987.

[Pur98] Anuj Puri. Dynamical properties of timed automata. In **FTRTFT**, pages 210–227, 1998.

[Ras99] Jean-François Raskin. **Logics, Automata and Classical Theories for Deciding Real Time**. Phd thesis, FUNDP University, Belgium, 1999.

[Ras05] Jean-François Raskin. An introduction to hybrid automata. In **Handbook of Networked and Embedded Control Systems**, pages 491–518. Birkhäuser, 2005.

[Ray15] Michel Raynal. Parallel computing vs. distributed computing: A great confusion? (position paper). In **Euro-Par 2015: Parallel Processing Workshops - Euro-Par 2015 International Workshops**, 2015.

[RDS$^+$15] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. Reactive synthesis from signal temporal logic specifications. In **Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control (HSCC 2015)**, pages 239–248, April 2015.

[RMS16] Suman Roy, Janardan Misra, and Indranil Saha. A simplification of a real-time verification problem. **Softw. Test. Verification Reliab.**, 26(8):548–571, 2016.

[RS97] Jean-François Raskin and Pierre-Yves Schobbens. State clock logic: A decidable real-time logic. In **HART**, pages 33–47, 1997.

[Saf89] Shmuel Safra. **Complexity of Automata on Infinite Objects**. phd thesis, The Weizmann Institute of Science, Israel, 1989.

[San13] Ocan Sankur. Shrinktech: A tool for the robustness analysis of timed automata. In **CAV**, volume 8044 of **Lecture Notes in Computer Science**, pages 1006–1012. Springer, 2013.

[San15] Ocan Sankur. Symbolic quantitative robustness analysis of timed automata. In **TACAS**, volume 9035 of **Lecture Notes in Computer Science**, pages 484–498. Springer, 2015.

[SBM14] Ocan Sankur, Patricia Bouyer, and Nicolas Markey. Shrinking timed automata. **Inf. Comput.**, 234:107–132, 2014.

[SC85] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. **In ACM**, pages 733–749, 1985.

[Sch99] Steve Schneider. **Concurrent and Real Time Systems: The CSP Approach**. John Wiley & Sons, Inc., New York, NY, USA, 1999.

[SDJ$^+$92] Steve Schneider, Jim Davies, D. M. Jackson, George M. Reed, Joy N. Reed, and A. W. Roscoe. Timed CSP: Theory and practice. In **Theory in Practice, REX Workshop**, pages 640–675. Universiteit Twente, 1992.

[SFK08] Mani Swaminathan, Martin Fränzle, and Joost-Pieter Katoen. The surprising robustness of (closed) timed automata against clock-drift. In **IFIP TCS**, volume 273 of **IFIP**, pages 537–553. Springer, 2008.

[SG11] Neda Saeedloei and Gopal Gupta. An extension of π-calculus with real-time and its realization in logic programming. In **Handbook of Networked and Embedded Control Systems**, pages 119–135, Cham, 2011. Springer International Publishing.

[Sif01] Joseph Sifakis. Modeling real-time systems-challenges and work directions. In **EMSOFT'01**, pages 373–389, 2001.

[Sti87] Colin Stirling. Comparing linear and branching time temporal logics. In **Temporal Logic in Specification**, pages 1–20, 1987.

[SVW85] A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for büchi automata with applications to temporal logic (extended abstract). In **Colloquium on Automata, Languages and Programming**, pages 465–474, 1985.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In **Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)**, pages 133–192. ACM, New York, NY, USA, 1990.

[Tho97] Wolfgang Thomas. **Languages, automata, and logic**, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[Tho02] Wolfgang Thomas. Infinite games and verification (extended abstract of a tutorial). In **CAV'02**, volume 2404, pages 58–64, 2002.

[Tri98] S. Tripakis. **The analysis of timed systems in practice**. PhD thesis, Université Joseph Fourier, Grenoble, 1998.

[TS06] Andrew S. Tanenbaum and Maarten van Steen. **Distributed Systems: Principles and Paradigms (2nd Edition)**. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[TY01] Stavros Tripakis and Sergio Yovine. Analysis of timed systems using time-abstracting bisimulations. **In Formal Methods in System Design**, pages 25–68, 2001.

[UPP] The UPPAAL tool. Available at `http://www.uppaal.com/`.

[VM97] Jeffrey M. Voas and Gary McGraw. **Software Fault Injection: Inoculating Programs Against Errors**. John Wiley & Sons, Inc., 1997.

[VvdPGS97] J. P. M. Voeten, P.H.A. van der Putten, M.C.W. Geilen, and M. P. J. Stevens. Formal modelling of reactive hardware/software systems. In **ProRISC/IEEE'97, Utrecht : STW, Technology Foundation**, pages 663–670, 1997.

[Wil94] Thomas Wilke. Specifying timed state sequences in powerful decidable logics and timed automata. In **FTRTFT**, pages 694–715, 1994.

[Win93] Glynn Winskel. **The formal semantics of programming languages: an introduction**. MIT Press, Cambridge, MA, USA, 1993.

[WL97] Carsten Weise and Dirk Lenzkes. Efficient scaling-invariant checking of timed bisimulation. In **STACS**, pages 177–188, 1997.

[WME93] Farn Wang, Aloysius K. Mok, and E. Allen Emerson. Distributed real-time system specification and verification in aptl. **ACM Trans. Softw. Eng. Methodol.**, volume 2, 1993.

[Wol83] Pierre Wolper. Temporal logic can be more expressive. **In Information and Control**, page 7299, 1983.