Model Checking for Time Division Multiple Access Systems

– PhD. Thesis –

Marco Muñiz

13th November 2014

Model Checking for Time Division Multiple Access Systems

zur Erlangung des Doktorgrads
der technischen Fakultät
der Albert-Ludwigs-Universität Freiburg im Breisgau
von

Marco Antonio Muñiz Rodríguez

13. November 2014

A mi familia.

Abstract

Software systems are present in everyday life. A large class of systems consist of components that communicate with each other using the Time Division Multiple Access (TDMA) paradigm. The presence of an error or an unexpected behavior in such a system can lead to severe consequences such as; the dead of people, catastrophic economical losses, etc. Therefore, the correct functioning of these systems is critical. Guaranteeing the correctness of a critical system is an active research area. A promising approach to ensure the correctness of critical systems is that of model checking using timed automata. Timed automata have been successfully used to model and verify many real world applications. However, in some cases the size of the state space induced by a timed automaton hinders the model checking task.

In this work, we contribute to the model checking of TDMA based systems. We present a number of techniques for reducing the state space induced by a timed automaton while preserving most or all of its properties. We formalize the underlying principles of TDMA based systems with the semantic based notions of disjoint activity, sequentialisability, and concatenation. We observe that the application of the corresponding techniques to TDMA systems, yield a transition system which is bisimilar to the one obtained by parallel composition. We achieve a quadratic speed up in the verification times in the number of components. We investigate the applicability of timed model checking to industrial systems. Towards this end, we provide syntactic patterns and operators for modeling TDMA based systems using timed automata. By applying these patterns, we obtain a transition system which is weakly bisimilar to the one obtained by parallel composition. We observe an improvement in the verification times from exponential to linear in the number of components. Finally, we address another aspect of real timed systems, i.e. clock differences of zero time duration introduced by the interleaving semantics of timed automata and clock resets. We call these clocks quasi-equal clocks. These clocks can be simplified, yielding a much smaller system. Before a simplification can take place, the presence of quasi-equal clocks need to be detected. Therefore, we provide an efficient abstraction method for detecting quasi-equal and equal clocks.

As a proof of concept, we have implemented our methods in our tool sAsEt and executed a number of experiments. Our experiments show the practical potential of the techniques presented in this thesis. In addition, the aforementioned methods have been used to model and detect design errors in a real world TDMA based system. We successfully verified an improved design of the real world TDMA system by using our methods.

Acknowledgements

I would like to thank my family whose support and encouragement made it possible for me to come to Germany and pursue my dreams of becoming a scientist.

I thank Andreas Podelski for accepting me as his student and for his support in the last eight years. I had the fortune of studying and working at his team under his supervision. Andreas refined taste for elegance and simplicity have been a source of inspiration and improvement.

I thank Bernd Westphal for his teachings, supervision and guidance during my masters and doctoral studies. I have enjoyed our many disscusions on new research topics and search for suitable solutions. I thank Bernd for encouraging me in broading my perception and in taking initiative.

I thank Jochen Hoenicke for the many disscusions, explanations and questions he answered during my masters and doctoral studies.

I thank Viktor Kuncak for hosting me as an intern at EPFL. His passion for science made the internship at his group an impresive and motivating experience. In addition, I thank Viktor for providing the Jahob verification system on top of which we have implemented our model checker sAsEt.

I thank Thomas Wies for introducing me to the area of theorem proving, for supervising me during my master studies and for supervising me during my internship at EPFL.

Finally, for all the nice experiences and pleasant atmosphrere, I thank all members of the Software Engineering group at the University of Freiburg, including: Andreas Dereani, Alexander Malkis, Alexander Nutz, Amalinda Post, Berit Brauer, Christian Herrera, Christian Schilling, Corina Mitrohin, Daniel Dietsch, Evren Ermis, Jeremi Dzienian, Jürgen Christ, Marius Greitschus, Marlis Jost, Martin Mehlman, Martin Schäf, Martin Wehrle, Martin Preen, Matthias Heizmann, Mohammed Nassim Seghir, Rebecca Albrecht, Sergio Alejandro Feo-Arenis, Sergiy Bogomolov and Stefan Maus. In particular, I thank Christian Herrera and Jeremi for helping me with the development of our tool and execution of experiments. I thank Sergio and Sergiy for comments on a preliminary version of this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Information systems are present in everyday life. These systems perform control tasks in trains, airplanes, cellphones, etc. In many cases the correct functioning of a system is critical. As an example, consider a system controlling a pacemaker. Unexpected behavior of this system could cost the life of a human being. Errors in information systems can not only cost human lives but also can lead to catastrophic economical losses. In the early nineties, an error in the Pentium II processor caused a loss of over four hundred million dollars to Intel. Unfortunately, in the last decades there have been numerous spectacular software and hardware failures. A large number of safety critical systems, implement the Time Division Multiple Access (TDMA [116]) paradigm. Systems based on a TDMA paradigm may control satellite communications, steering and breaking of modern vehicles, security systems, etc. Since the importance and use of information systems is increasing, ensuring their correct functioning is a prime concern.

Ensuring correctness of a system is desirable. However, this is not an easy task. There are several approaches to remove errors from systems. We briefly discuss four major approaches for improving the correct functioning of systems. These approaches are: testing [102, 133], simulation [93, 35], deductive verification [73, 104], and model checking [45, 115]. First, we consider testing. In testing, a finite number of scenarios is executed in a concrete system. If an error occurs, then the concrete system can be modified. Second, we consider simulation. In simulation, a finite number of possible scenarios is simulated in an abstract system or model. If an error occurs, then the model can be refined. Third, we consider deductive verification. In deductive verification, specifications of the expected behavior of the system are provided. The system and its specifications are then decomposed in logical entailments which are proven for validity using a theorem prover. Finally, we consider model checking. In model checking, the specifications are checked algorithmically to hold in a model corresponding to a concrete system. If an error is detected, then the model can be refined. While in testing and simulation only a finite number of possible scenarios are explored, deductive verification and model checking are exhaustive, i.e. they explore all possible scenarios. An important difference between deductive verification and model checking is that model checking is fully automatic. That is, the verification process does not require the aid of an expert. This difference and recent advances make model checking an attractive and applicable approach to ensure the correctness of critical systems.

In order to formally verify a system using model checking. Two components are

required. First, a formal model describing the behavior of the system. Second, a number of specifications in a formal language. A prominent theory for modeling real time systems is the theory of timed automata [5]. Timed automata have been successfully used to model and verify many real world applications.

An important characteristic of systems based on the Time Division Multiple Access paradigm is that they are real time systems. Therefore, the use of timed automata as models for TDMA systems is a natural choice. In industrial applications TDMA based systems involve a large number of components. Modeling such industrial applications using timed automata will thus involve a large number of automata. Reasoning about such systems in general requieres a composition of the components. A common problem in model checking of such models, is the so called state explosion problem [129], i.e. the state space grows exponentially in the number of components. Thus, using standard model checking techniques on industrial TDMA based systems often does not scale.

In this work, we contribute to the model checking of Time Division Multiple Access based systems. In general, we present a number of techniques for avoiding the state explosion problem. By using these techniques we have been able to detect errors, correct errors and successfully verify a real world TDMA based system [58].

We now continue with an outline for the rest of this chapter. In Section 1.1, we present an overview of model checking. In Section 1.2, we briefly present the TDMA paradigm. In Section 1.3, we give an example on how to model a TDMA based system using timed automata. In Section 1.4, we present our technical contributions. In Section 1.5, we provide an outline for the rest of this thesis.

1.1 Model Checking

Model checking [45, 115] is an automated technique that, given a finite-state model of a system and a formal property, algorithmically checks whether the property holds for that model. There are several components and steps in the model checking process. Figure 1.1 describes the process. We briefly discuss the main elements of the model checking process.

System

In the model checking process, a concrete system is provided. This system can be a communication protocol, a hardware specification, software program, etc. The system is usually specified in programming languages like C, Java, or hardware description languages such as Verilog [126] or VHDL [103].

Abstract Model

The system needs to be represented in terms of the description language of the model checker. The system represented in the description language is the abstract model of the system. The abstract model describes the system behavior in a mathematically precise

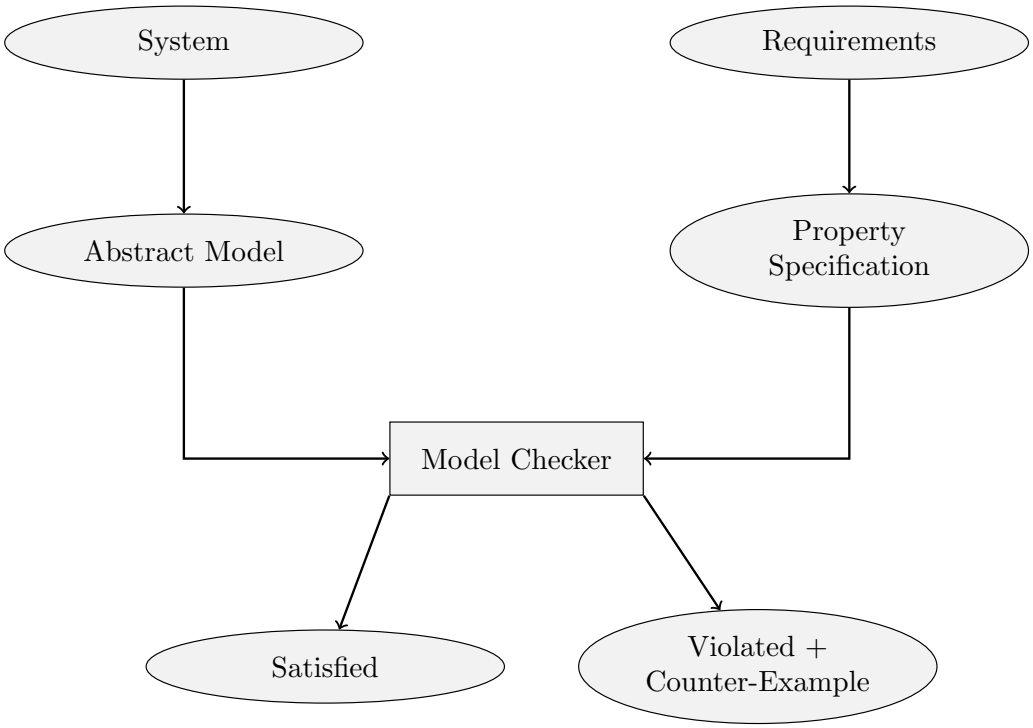


Figure 1.1: Model checking process

and unambiguous manner. In this work, we use timed automata as a description language. The abstract model can be automatically obtained from the system description, provided the existence of a procedure for transforming the system into a formal description. If such an automatic procedure does not exist, a modeler can perform this step manually.

Requirements

Requirements describe properties of the system. Requirements specify what the system should or should not do. As an example, consider an elevator. A requirement could specify that the doors can only be opened if and only if the elevator is stopped.

Formal Specification

In order to formally verify a system, requirements should be described in a property language. In model checking, properties are generally specified using *temporal logic* [113, 41]. In this work, we use Timed Computation Tree Logic(TCTL) [2] as specification language.

Model Checker

The model checker has as input the transition system induced by the abstract model and a property specification. The model checker then systematically explores the transition system checking for validity of the property. There are three possible outcomes:

1. The property is satisfied. In this case the model checking process has been successful and the transition system satisfies the property.

2. The property is violated. In this case the model does not satisfy the specification and the model checker provides a counter example. With help of the counter example the system can be refined excluding the detected error. A new iteration is needed to verify the refined model.

3. The model checker goes out of memory. This is because the size of the transition system is too big to be represented in a physical computer. A solution is to reduce the size of the model and try again. In this thesis we focus on techniques for avoiding this case.

Model checking is a "push-button" technology. This means that the model checking process does not need interaction from the user. In addition, the user does not need to be an expert. Finally, the use of model checking can effectively increase the reliability of a system.

1.2 Time Division Multiple Access (TDMA)

Time Division Multiple Access (TDMA [116]) is a *channel access method* (CAM) used to facilitate channel sharing without interference. TDMA allows multiple nodes to share and use the same transmission channel by dividing signals into different time slots. Users transmit in rapid succession, and each one uses its own time slot. Thus, multiple stations may share the same frequency channel but only use part of its capacity.

Frames and Slots

Figure 1.2 describes the structure of the TDMA paradigm. The information is transmitted in so called frames. The size of a frame is given by a fixed number of time units. We also refer to frames as cycles. Every frame is divided into time slots. The number of time slots is constant in every frame. The size of a time slot is obtained by dividing the size of a frame by the number of slots it has. Real world applications use physical clocks. Since a TDMA based system consist of several independent components. The system will involve several clocks. Because the precision of physical clocks is limited, clocks will progress at is own pace. Clocks may thus not be synchronized an may cause problems in the TDMA cycle. This phenomena is known as clock drift. The TDMA paradigm addresses this issue by assigning time guards to the time slots. By choosing right time guards and using accurate clocks, perfect clocks can be assumed [88]. Perfect clocks are

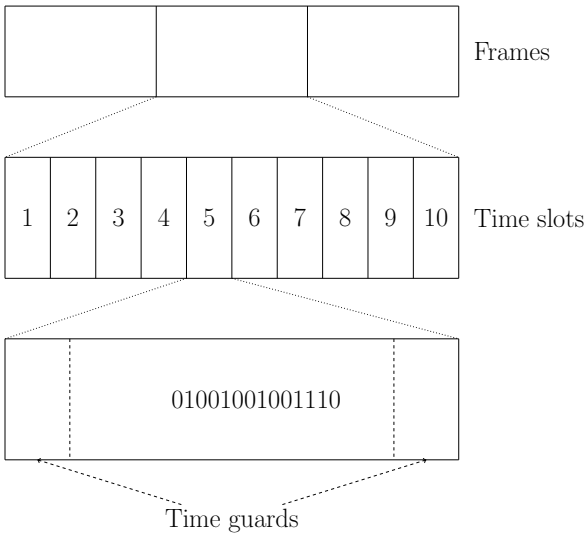


Figure 1.2: TDMA structure, The data to be transmitted will be split into Frames. Every frame is split into time slots. The Information is transmitted in the corresponding time slot.

important for our work since they allow us to abstract from the problems induced by clock drift.

TDMA Based Protocols

A great number of protocols are based on the TDMA paradigm. TDMA is used in the digital 2G cellular systems such as Global System for Mobile Communications (GSM), IS-136, Personal Digital Cellular (PDC), and in the Digital Enhanced Cordless Telecommunications (DECT) standard for portable phones. It is also used extensively in satellite systems, combat-net radio systems, and Passive Optical Network (PON) for upstream traffic from premises to the operator.

The TDMA paradigm is not only used in wireless systems, but also in LAN networks. The ITU-T G.hn standard, which provides high-speed local area networking over existing home wiring (power lines, phone lines and coaxial cables) is based on a TDMA scheme. In G.hn, a "master" device allocates "Contention-Free Transmission Opportunities" (CFTXOP) to other "slave" devices in the network. Only one device can use a CFTXOP at a time, thus avoiding collisions. The FlexRay protocol, which is also a wired network protocol used for safety-critical communication in modern cars, uses the TDMA method for data transmission control.

Another important protocol for safety critical applications is the Timed Triggered Protocol (TTP). Timed Triggered Protocol is often used in mission critical data communications applications, such as aircraft engine management and other aerospace ap-

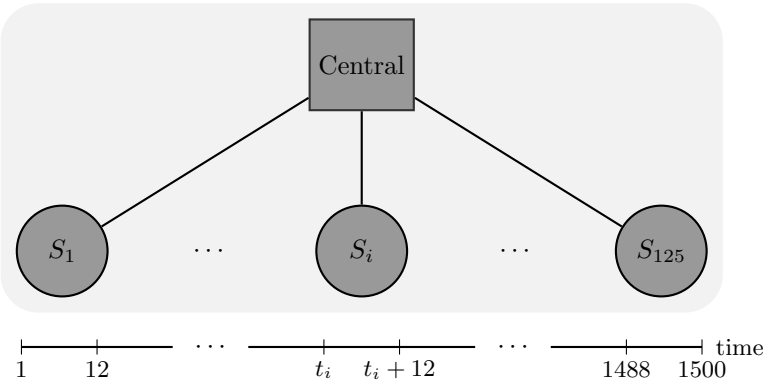


Figure 1.3: The fire alarm system consisting of a central unit and a 125 sensors. The 150 seconds cycle is divided in 125 slots. Every sensor is assigned to a slot. Sensors can only communicate with the central unit. The sensors and the central unit are connected in a wireless network.

plications where deterministic operation is a requirement. In these applications, the TTP networks are often operated as separate networks with separate communication controller interface devices and separate, but coordinated, configurations.

1.3 Case Study: Fire Alarm System

In the following, we present a fire alarm system which is based on the TDMA paradigm. This system is a simplification of a real world system [58]. We have successfully verified the corresponding real world fire alarm system using, among other techniques, the techniques presented in this thesis. We will use the simplified fire alarm system to illustrate several techniques presented in this thesis.

The fire alarm system consists of a central unit and 125 sensors. The communication protocol of the fire alarm system is based on the TDMA paradigm. That is, there exists a cycle of 150 seconds. The cycle is divided into 125 slots. For modeling we use deciseconds, i.e. the cycle has a length of 1500 units. Therefore, every slot has a duration of 12 time units. Every sensor is assigned to a slot. Sensors can not communicate with each other. They only communicate with the central unit. The components of the fire alarm system are connected in a wireless network. Figure 1.3 illustrates the simplified fire alarm system that we consider.

The fire alarm system has several modes. We consider the surveillance mode. In this mode, the central unit monitors the correct functioning of the sensors. At every slot, the central unit awaits for an alive message of the corresponding sensor. If the alive message is received, it replies with an acknowledgment. If the alive message is not received, it shows an error.

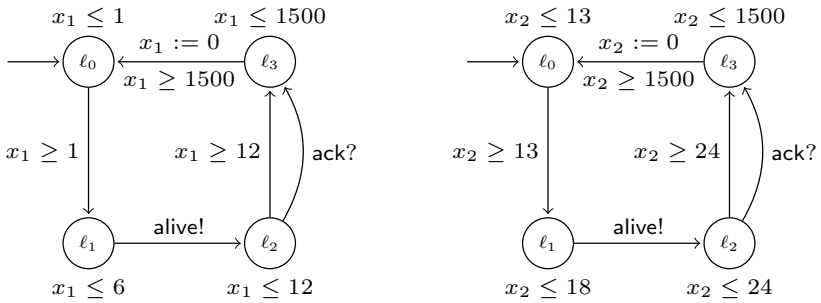


Figure 1.4: Modeling the system with timed automata, sensors.

Remark 1.1. The fire alarm system we have presented is a simplified one. We have verified the corresponding real world fire alarm system. This system has up to ten repeaters and implements complex fault tolerance mechanisms. For the sake of simplicity in this thesis, we abstract from many features of the real system. The techniques presented in this thesis have been successfully applied to the real world fire alarm system. If the reader is interested in the modeling and formalization of requirements for the real fire alarm system, this can be found in [58]. ♢

The Model

We use timed automata to model the fire alarm system. Figure 1.4 shows timed automata corresponding to sensor 1 and sensor 2. Figure 1.5 shows the model for the central unit. All sensors function in a similar way. Therefore, their structure is quite similar. The only differences are the time constrains imposed by the time slot they are assigned to. We briefly discuss the models corresponding to the sensors and the central unit. As a time unit for the model we use deciseconds. Therefore, the TDMA cycle consists of 1500 time units.

Sensors The automaton at the left of Figure 1.4 describes the behavior of sensor 1. Sensor 1 is assigned to slot 1. Slot 1 ranges from time unit 1 to time unit 12. The model for sensor 1 has four locations $\ell_0, \ell_1, \ell_2, \ell_3$ and one clock x_1. The behavior of sensor 1 is as follows: At location ℓ_0 sensor 1 awaits the start of its slot. This is modeled by the location invariant $\ell_0 \leq 1$. At point of time 1, the sensor enters its time slot and moves to location ℓ_1. At location ℓ_1 the sensor can send his alive message non-deterministically at any point in time in the interval from 1 to 6. This is modeled by the location invariant $x_1 \leq 6$. After sending its alive message, the sensor moves to location ℓ_2. At this location, the sensor awaits an acknowledgment from the central unit. If an acknowledgment is received or the end of the time slot is reached, the sensor moves to ℓ_3. At this location, the sensor waits until the end of the TDMA cycle, i.e. 1500 time units. At the end of the

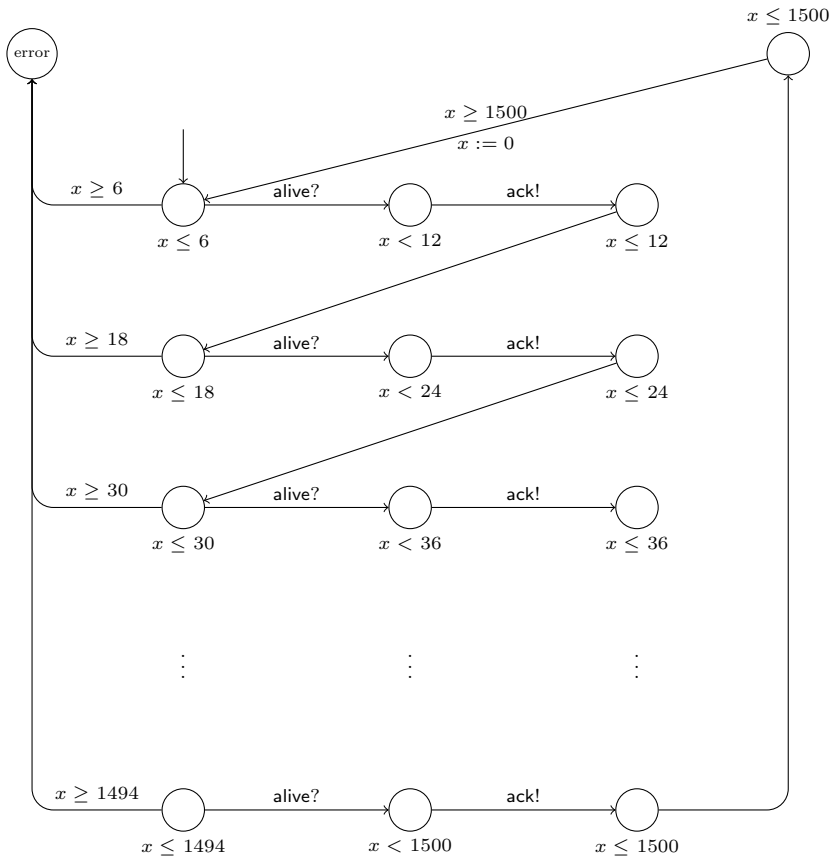


Figure 1.5: Model-ling the system with timed automata, central unit

cycle, the sensor resets its clock x_1 and moves to location ℓ_1. All sensors are modeled in an analogous way.

Central unit The automaton at Figure 1.5 describes the behavior of the central unit. The automaton has 125 layers, each layer corresponds to the communication of the central unit with a sensor. Every layer functions in a similar way. We explain the first layer. In the first layer, the central unit waits until point of time 6 for an alive message from sensor 1. This is expressed by the location invariant $x \leq 6$. If the alive message has not been received until the point of time 6, the central unit moves to the error location. If the alive message has been received, the central unit moves to the next location and replies with an acknowledgment. At the end of the first slot, the model moves to the next location at layer two and communicates with sensor number 2. At the end of the TDMA cycle, the central unit resets its clock and returns to layer one to communicate with the first sensor.

The Properties

There are several properties relevant to the real fire alarm system. The system is a *reactive system*. Therefore, the corresponding model should satisfy the *deadlock freedom* property. Because of European standards, another property might be, that if a sensor is not working, this should be detected and displayed in less than 300 seconds. Wireless systems face interference problems. Hence, retransmissions of alive messages are important. A property may require, that under certain environmental circumstances alive messages are always received by the central. For the simplified fire alarm system, we may require the location error of the central unit to be unreachable.

The Verification Process

In the model for the fire alarm system presented above, the models describing the sensors correspond to an obvious design. By this we mean that, since sensors are independent components, it is a natural way to model every sensor with one clock.

In the fire alarm system, most of the properties that we need to verify are *invariant properties* . Invariant properties require the complete state space to be checked. In order to analyze the behavior of the fire alarm system, the components need to be composed. Using standard operators, such as the parallel composition on the 125 sensors, will render the verification intractable. This is because, the size of the resulting system grows exponentially in the number of sensors.

A closer look at the principles of TDMA systems reveals important properties that we exploit to avoid the state explosion problem. As an example, in the fire alarm system, we observe that sensors operate in different time slots. This leads to the notion of *disjoint activity*. In Chapter 3, we identify and formalize several underlying ideas of TDMA based systems. In Chapter 4, we contribute to the verification of TDMA based systems by providing syntactic patterns and operators. By using the techniques presented in Chapter 4, the fire alarm system presented can be verified in linear time on the number of sensors. Another important observation is that the clocks of the sensors are almost equal. In the fire alarm system, at point of time which are multiples of 1500, it can be the case that clock $x_1 = 1500$ and clock $x_2 = 0$. These clocks are not equal, but *quasi-equal*. This clocks can be simplified. In Chapter 5, we present an abstraction method for efficiently detecting quasi-equal clocks.

1.4 Summary of Contributions

In summary, this dissertation makes the following contributions:

- We provide a formalization of important properties of TDMA based systems. We introduce concepts such as *periodic cyclic timed automata*, *disjoint activity*, *sequentialiasability* and *concatenation*. We show that the concatenation of sequentialisable timed automata yields a bisimilar system with respect to the one obtained

using parallel composition. However, with a reduced number of edges. We show that application of these methods lead to quadratic speed ups.

- We introduce a new syntactic class of timed automata. We call this class *sequential timed automata*. This class of automata enables the syntactical check of properties relevant to disjoint activity and sequentialiasability. Sequential timed automata are well suited to model the components of TDMA based systems. Next, we introduce the notion of *overclocks* and *sequential composition* on sequential timed automata. We show the relation between a system produced by using the parallel composition operator and a system produced using the sequential composition operator. This relation is a weak bisimulation, thus preserving most properties. Applying the sequential operator reduces the time complexity, for the model checking task, from exponential to linear in the number of components.

- We detect another important component of real timed systems i.e. clock differences of zero time duration. The clock differences are introduced by the interleaving semantics of timed automata and clock resets. We call these clocks *quasi-equal clocks*. Quasi-equal clocks can be replaced by a representative clock, yielding a much smaller system. In order to perform a reduction on the number of clocks, the quasi-equal clocks have to be efficiently detected. Therefore, we introduce the notions of *zero time configurations*, *zero time behavior*, and a *relax operator* on zones. By using these notions, we construct an *abstract zone graph*. In this graph, quasi-equal and equal clocks can be efficiently detected. We provide upper bounds for the size of the abstract zone graph, and formalize its relation to the corresponding concrete zone graph. Finally, we provide an algorithm for computing the abstract zone graph.

- We present our tool sAsEt. A flexible tool for analyzing timed systems. sAsEt is based on the Jahob system. sAsEt encodes regions and zones as Isabelle/HOL formulae. This allows our tool to use several theorem provers as constraint solvers.

The techniques above have been applied to verify a real world fire alarm system. A detailed description of the verification process of this system can be found in [58]. Notions given above corresponding to concatenation and sequentialisation of timed automata have been presented in [100]. Notions given above corresponding to quasi-equal clocks have been presented in [76, 101]. In addition, we have been able to successfully extend our work on quasi-equal clocks in timed automata to *quasi-dependent variables* in hybrid systems. These work has been presented in [30]. We have used our tool sAsEt to compute the results presented in [30, 101].

1.5 Outline

In Chapter 2, we give the preliminaries of this thesis. We briefly introduce timed automata and Timed Computation Tree Logic (TCTL). In Chapter 3, we study and formalize

by means of timed automata important underlying principles of TDMA based systems. Chapter 4 presents syntactic patterns and operators. These patterns are well suited for the modeling and verification of TDMA based systems. The use of these patterns render the verification complexity from exponential to linear in the number of components. Chapter 5 presents the notion of quasi-equal clocks and an abstraction method for efficiently detecting such clocks. In Chapter 6, we present our tool SAsEt and benchmarks obtained by using the techniques presented in this thesis. In Chapter 7, we present our conclusions and future work. Finally, Chapter 8 present a summary of this thesis in the German language.

Chapter 2

Preliminaries

Contents

This chapter presents important notions and notation that we will use in the rest of this thesis. Section 2.1 presents key principles of the theory of timed automata. A finite abstraction method for timed automata is presented. Section 2.2 presents the syntax and semantics of a fragment of Timed Computation Tree Logic.

2.1 Timed Automata

The formal basis for our work are *timed automata* [5], our presentation follows [27, 105]. Timed automata are useful for modeling the behavior of real time critical systems. They allow to describe relevant timed properties about the behavior of a system.

Timed automata are finite automata equipped with a finite set of real-valued variables. These variables are called *clocks*. Clocks indicate the time elapsing at a certain location. Clocks can be inspected or be reset to zero. The modeled system may require that a location is visited for a limited number of time units. For this, *clock constraints* are needed. The formal definition of clock constraints follows.

Definition 2.1 (Clock Constraints)**.** *Let* $\mathbb{X}$ *be a set of clocks. The set* $\Phi(\mathbb{X})$ *of simple clock constraints over* $\mathbb{X}$ *is defined by the following grammar*

$$\varphi ::= x \sim c \mid x - y \sim C \mid \varphi_1 \wedge \varphi_2$$

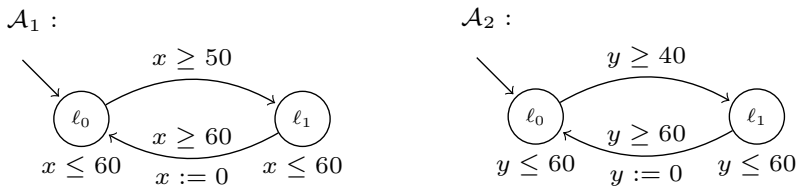


Figure 2.1: Example of two timed automata.

where $x, y \in \mathbb{X}$, $C \in \mathbb{Q}_0^+$, *and* $\sim \in \{<, \leq, \geq, >\}$. *Constraints of the form* $x - y \sim c$ *are called* difference constraints. ♢

Intuitively, a timed automaton is a directed graph with variables (clocks). The clocks are used to model the system behavior. The locations in a timed automaton have a clock constraint. Such a clock constraint is called *location invariant*. A location invariant indicates the time the automaton can spend at the given location. The edges in a timed automaton have *guards* and *clock resets*. Guards are clock constraints which indicate at which points of time the edge is "enable" or it can be taken. Clock resets assign the value of zero to a number of clocks. The formal definition of timed automaton follows.

Definition 2.2 (Timed Automaton)**.** *A Timed automaton* $\mathcal{A}$ *is a tuple*

$$(L, \Sigma, \mathbb{X}, I, E, \ell_0),$$

where

- L *is a finite set of* locations*, with typical element* ℓ*.*
- Σ *is a finite set of* actions *comprising the* internal action τ*.*
- $\mathbb{X}$ *is a finite set of* clocks*.*
- $I : L \to \Phi(\mathbb{X})$ *is a mapping that assigns to each location a* clock constraint*, its invariant.*
- $E \subseteq L \times \Sigma \times \Phi(\mathbb{X}) \times \mathcal{P}(\mathbb{X}) \times L$ *is the set of* edges*. An edge* $e = (\ell, \alpha, \varphi, Y, \ell') \in E$ *from* ℓ *to* ℓ' *involves an action* $\alpha \in \Sigma$*, a* guard $\varphi \in \Phi(\mathbb{X})$*, and a* reset set $Y \subseteq \mathbb{X}$*.*
- $\ell_0 \in L$ *is the* initial location*.* ♢

Example 2.3. Figure 2.1 depicts two timed automata. Timed automaton $\mathcal{A}_1$ and timed automaton $\mathcal{A}_2$. In timed automaton $\mathcal{A}_1$ the set of locations L consists of locations ℓ_0

and ℓ_1. The set of actions Σ consist of the internal action τ. The set of clocks $\mathbb{X}$ is a singleton consisting of clock x. The invariant function assigns to every location, the constraint that the clock has to be less or equal than 60. The edge going from ℓ_0 to ℓ_1 has a guard stating, that it can only be taken if the value of the clock x is greater or equal than 50. The edge from location ℓ_1 to ℓ_0 has a reset setting the value of the clock x to zero. The initial location is ℓ_0. ♢

Complex systems consist of several components where the components operate concurrently. One way to analyze the interaction among these components is by composing all them using a *parallel product* operator. Intuitively, the parallel product of two timed automata will produce a timed automaton which syntactically considers all possible scenarios. Note, that the parallel product applied on two timed automata, computes the Cartesian product on the set of locations. The formal definition of the parallel product operator follows.

Definition 2.4 (Parallel Product)**.** *The* parallel product *of two timed automata*

$$\mathcal{A}_i = (L_i, \Sigma_i, \mathbb{X}_i, I_i, E_i, \ell_{0,i}) \text{ with } i = 1, 2,$$

and disjoint sets of clocks $\mathbb{X}_1$ *and* $\mathbb{X}_2$ *yields the timed automaton*

$$\mathcal{A}_1 \| \mathcal{A}_2 \stackrel{\text{def}}{=} (L_1 \times L_2, \Sigma_1 \cup \Sigma_2, \mathbb{X}_1 \cup \mathbb{X}_2, I, E, (\ell_{0,1}, \ell_{0,2}))$$

where :

- $I(\ell_1, \ell_2) := I_1(\ell_1) \wedge I_2(\ell_2)$*, for each* $\ell_1 \in L_1, \ell_2 \in L_2$*,*
- *and* E *consists of* handshake *and* asynchronous edges *where:*
  - *There is a handshake transition* $((\ell_1, \ell_2), \tau, \varphi_1 \wedge \varphi_2, Y_1 \cup Y_2, (\ell_1', \ell_2')) \in E$ *if there are* complementary *actions* α *and* $\bar{\alpha}$ *in* $\Sigma_1 \cup \Sigma_2$ *such that* $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ *and* $(\ell_2, \bar{\alpha}, \varphi_2, Y_2, \ell_2') \in E_2$*.*
  - *There is an asynchronous transition* $((\ell_1, \ell_2), \alpha, \varphi_1, Y_1, (\ell_1', \ell_2)) \in E$*, for each edge* $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ *and each location* $\ell_2 \in L_2$*, and analogously for each transition in* E_2*.* ♢

Example 2.5. Figure 2.2 shows the timed automaton resulting from applying the parallel product operator to timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ from Figure 2.1. The initial location is a pair consisting of the corresponding initial locations. The location invariants are conjunctions of the corresponding location invariants in $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Note that all possible location and edge combinations are present in the resulting timed automaton. ♢

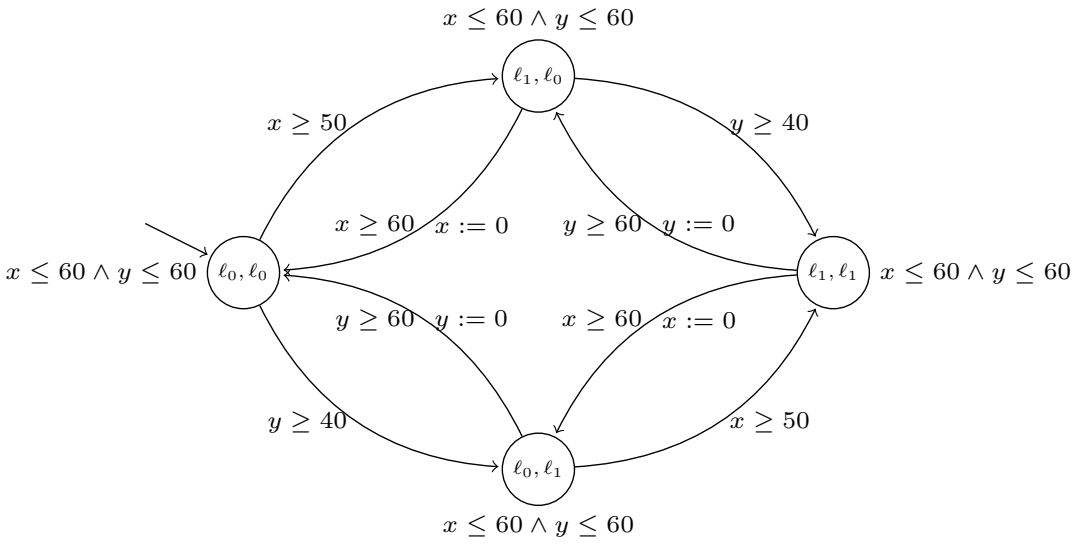


Figure 2.2: Timed automaton resulting from the parallel product of timed automaton $\mathcal{A}_1$ and timed automaton $\mathcal{A}_2$ from Figure 2.1

2.1.1 Semantics

From previous examples we observe, that the current state of a timed automaton is given by the current location and its current clock values. The clock values are formalized using *valuations*. Therefore, the states or *configurations* of a timed automaton are pairs consisting of locations and valuations. The edges in a timed automaton induce a *transition relation* between configurations. Therefore, the underlying semantics of a timed automaton is given by an infinite system. These systems are know as infinite *transition systems*. In the following we formalize these notions. For the rest of this section, let us fix a timed automaton $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$.

Definition 2.6 (Valuation)**.** *A* valuation ν *of clocks in* $\mathbb{X}$ *is a mapping*

$$\nu : \mathbb{X} \to \mathbb{R}_0^+,$$

assigning to each clock $x \in \mathbb{X}$ *the current time.* ♢

In order to evaluate if a guard is satisfied or if the location invariant is still satisfied by a valuation, we need to define a *satisfaction relation* between valuations and clock constraints.

Definition 2.7 (Satisfaction Relation)**.** *The satisfaction relation "*$\models$*" between valu-*

ations and clock constraints is defined inductively

$$\begin{aligned} \nu \models x \sim c \quad & \textit{iff} \quad \nu(x) \sim c, \\ \nu \models x - y \sim c \quad & \textit{iff} \quad \nu(x) - \nu(y) \sim c, \\ \nu \models \varphi_1 \wedge \varphi_2 \quad & \textit{iff} \quad \nu \models \varphi_1 \textit{ and } \nu \models \varphi_2. \end{aligned}$$

If valuation ν satisfies the clock constraint φ we write

$$\nu \models \varphi.$$

$\diamondsuit$

The operational semantics of a timed automaton are given by a labeled transition system. By being at a location, a timed automaton can proceed in two ways. It can take an edge or it can let time progress while staying in that location. In the former, the system performs *discrete transition*, and the transition is labeled by an action. In the latter, the systems performs a *delay transition* and the transition is labeled by a positive real number indicating the elapsed time.

Definition 2.8 (Operational Semantics)**.** *The* operational semantics *of the timed automaton $\mathcal{A}$ is the* labeled transition system

$$\mathcal{TS}(\mathcal{A}) = (\mathit{Conf}(\mathcal{A}), \mathbb{R} \cup \Sigma, \{\xrightarrow{\lambda} \mid \lambda \in \mathbb{R} \cup \Sigma\}, C_0),$$

where:

- $\mathit{Conf}(\mathcal{A}) = \{(\langle \ell, \nu \rangle, t) \in L \times (\mathbb{X} \to \mathbb{R}) \times \mathbb{R} \mid \nu \models I(\ell)\}$ *is the set of* configurations *consisting of time-stamped pairs of a location $\ell \in L$ and a valuation of the clocks $\nu : \mathbb{X} \to \mathbb{R}$ which satisfies the clock constraint $I(\ell)$.*

- $\mathbb{R} \cup \Sigma$ *is the set containing all labels that may appear at transitions.*

- $\xrightarrow{\lambda} \subseteq \mathit{Conf}(\mathcal{A}) \times \mathit{Conf}(\mathcal{A})$ *is the* transition relation *where:*

  - *There is a* delay transition *from configuration $\langle \ell, \nu \rangle, t$ to $\langle \ell, \nu + t' \rangle, t + t'$. Formally*

    $$\langle \ell, \nu \rangle, t \xrightarrow{t'} \langle \ell, \nu + t' \rangle, t + t'$$

    if and only if $\nu + t'' \models I(\ell)$ for all $t'' \in [0, t']$, where $\nu + t'$ denotes the valuation obtained from ν by time shift t'.

– *There is an* action transition *between* $\langle \ell, \nu \rangle, t$ *and* $\langle \ell', \nu' \rangle, t$. *Formally*

$$\langle \ell, \nu \rangle, t \xrightarrow{\alpha} \langle \ell', \nu' \rangle, t$$

if and only if there exists an edge $(\ell, \alpha, \varphi, Y, \ell') \in E$ *with* $\nu \models \varphi$, $\nu' = \nu[Y := 0]$, *and* $\nu' \models I(\ell')$, *where* $\nu[Y := 0]$ *denotes the valuation obtained from* ν *by resetting exactly the clocks in* Y.

- $C_0 = \{(\langle \ell_0, \nu_0 \rangle, 0)\} \cap \mathit{Conf}(\mathcal{A})$ *is the set of initial configurations is where* $\nu_0(x) = 0$ *for all clocks* $x \in \mathbb{X}$.

♢

A timed automaton, starting at its initial configuration, by taking edges can move from one location to another one. If the clock constraints are satisfied. This process induces a *computation*. Every computation describe a possible behavior of the system.

Definition 2.9 (Computation)**.** *An infinite or maximally finite sequence*

$$\pi = c_0 \xrightarrow{\lambda_0} c_1 \xrightarrow{\lambda_1} c_2 \xrightarrow{\lambda_2} c_3 \ldots$$

is called a computation *of* $\mathcal{A}$ *if and only if* $c_0 \in C_0$ *and* $(c_i, c_{i+1}) \in \xrightarrow{\lambda}$ *for all* $i \in \mathbb{N}_0$. *We write* $\pi \in \mathcal{TS}(\mathcal{A})$ *if and only if* π *is a computation of* $\mathcal{A}$. ♢

Note that, in contrast to [105], we do not distinguish transition sequences and (time-stamped) computation paths. Here, the configurations of the labeled transition system are already time-stamped.

Definition 2.10 (Additional notation)**.** *Given a computation* $\pi \in \mathcal{TS}(\mathcal{A})$ *we write:*

- π_j *to denote the* j*-th configuration* $c_j = \langle \ell_j, \nu_j \rangle, t_j$ *in* π.
- λ_j^π *to denote the label of* j*-th transition in* π, *or simply* λ_j *if* π *is clear from the context.*
- $\ell(\pi_i)$, $\nu(\pi_i)$, *and* $t(\pi_i)$, *to denote the location* ℓ, *valuation* ν, *and time-stamp* t *of a configuration* $\langle \ell, \nu \rangle, t = \pi_i$.

♢

In the operational semantics, configurations are pairs consisting of a location and a valuation. Another equivalent description of the behavior of the system is know as *symbolic semantics* [18] . Symbolic semantics are useful for verification and presentation purposes. This is because they represent the behavior of the system in a more compact way. In the symbolic semantics configurations are pairs consisting of locations and clock constraints. The maximal set of valuations satisfying a clock constraint is known as a *zone*.

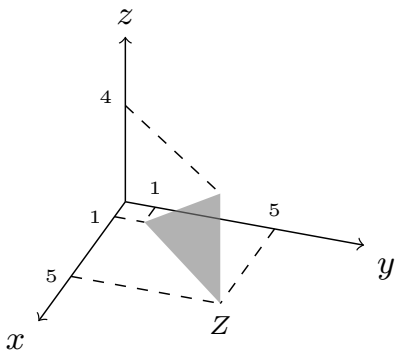


Figure 2.3: The zone for clock constraint $Z := x = y \wedge x \geq 1 \wedge y \leq 5 \wedge y - z \leq 1$

Definition 2.11 (Zone)**.** *A* zone *is the maximal set of clock valuations satisfying a clock constraint. Formally, let $Z \in \Phi(\mathbb{X})$ be a clock constraint then the induced zone $[Z]$ is*

$$[Z] = \{\nu \mid \nu \models Z\}$$

In the following we shall use Z to stand for $[Z]$ as a shorthand. Then, $\Phi(\mathbb{X})$ denotes the set of zones for $\mathcal{A}$. ♢

Example 2.12. Consider the clock constraint $Z := x - y \leq 0 \wedge x - y \geq 0 \wedge x \geq 1 \wedge y \leq 5 \wedge y - z \leq 1$. The zone $[Z]$ is the set of valuations satisfying the constraint Z. Figure 2.3 depicts the zone $[Z]$. ♢

At a given configuration, a timed automaton can perform a discrete transition or a delay transition. Since in the symbolic semantics configurations are pairs of locations and zones, a number of operations on zones are required to compute the successors of a configuration. For our purposes we only need to consider the effect of resets and time passage. Later, we introduce a normalization operator to ensure the finiteness of the system [61, 18].

Definition 2.13 (Operations on Zones)**.** *We define a* delay *operator and a* reset *operator on zones. Let $Z \in \Phi(\mathbb{X})$ and $Y \subseteq \mathbb{X}$, then*

- *A delay on zone Z is*
$$Z^{\uparrow} = \{\nu + d \mid \nu \in Z, d \in \mathbb{R}_+\}$$
- *A reset on zone Z for the clocks in Y is*
$$Z[Y := 0] = \{\nu[Y := 0] \mid \nu \in Z\}$$

where $\nu[Y := 0]$ *denotes the valuation obtained from* ν *by resetting exactly the clocks in* Y*.*

$\diamondsuit$

The delay operator applied to a zone extends it by adding all positive real numbers, i.e. it adds all the possible delay successors. Given a set of clocks, the reset operator applied to a zone, sets all the clocks in the reset set to zero.

The symbolic semantics are a more compact representation than the concrete semantics. In some fortunate cases, the symbolic semantics may yield a finite system. However, in general the symbolic semantics may yield an infinite system. The symbolic semantics for a timed automaton are given by a transition system whose configurations are pairs consisting of locations and zones. This transition system is known as *zone graph.*

Definition 2.14 (Zone Graph)**.** *The* symbolic semantics *for timed automaton* $\mathcal{A}$ *is defined by the* zone graph

$$ZG(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow_{ZG}, c_0)$$

where:

- $SConf(\mathcal{A}) = L \times \Phi(\mathbb{X})$ *is the set of configurations consisting of pairs of a location* $\ell \in L$ *and a zone* $Z \in \Phi(\mathbb{X})$*.*
- $c_0 = \langle \ell_0, \{\nu_0\} \rangle$ *is the initial configuration where* $\nu_0(x) = 0$ *for all clocks* $x \in \mathbb{X}$*.*
- $\rightarrow_{ZG} \subseteq SConf(\mathcal{A}) \times SConf(\mathcal{A})$ *is the transition relation where:*
  - *There is a delay transition from configuration* $\langle \ell, Z \rangle$ *to configuration* $\langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$*. Formally,*
    $$\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$$
  - *There is an action transition from configuration* $\langle \ell, Z \rangle$ *to configuration* $\langle \ell', (Z \wedge \varphi)[Y := 0] \wedge I(\ell') \rangle$*. Formally,*
    $$\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell', (Z \wedge \varphi)[Y := 0] \wedge I(\ell') \rangle$$
    if and only if there exists an edge $(\ell, \alpha, \varphi, Y, \ell') \in E$*.*

$\diamondsuit$

2.1.2 Finite Abstractions

Even though the symbolic semantics are a more compact representation than the concrete semantics. The symbolic semantics may yield an infinite system, which is not an adequate model for automated verification. In the following, we present an abstraction method for ensuring the resulting system to be finite. The foundation for the decidability results in timed automata is based in the notion of region equivalence over clock valuations [2, 5]. The equivalence relation is then used to construct a so called *region automata*. The region automaton is a finite system, which is bisimilar to the infinite transition system induced by the operational semantics. The region automaton is an important theoretical result. However, since the number of regions grows extremely fast, the construction of the region automaton is impractical. A more efficient representation of the state-space for timed automata is based on the notion of zones and zone-graphs [61, 75, 138].

Definition 2.15 (Equivalence Relation on Valuations)**.** *For timed automaton $\mathcal{A}$, let $\mathcal{G}$ be a finite set of difference constraints and $k : \mathbb{X} \to \mathbb{Q}_0^+$ be a function mapping each clock x to the maximal constant $k(x)$ appearing in the guards or invariants in $\mathcal{A}$ containing x. For a real d let $\{d\}$ denote the fractional part of d and $\lfloor d \rfloor$ denote its integer part. Two valuations ν, ν' are* equivalent*, denoted $\nu \stackrel{\cdot}{\sim}_{k,\mathcal{G}} \nu'$ iff*

1. *for all x, either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x) > k(x)$ and $\nu'(x) > k(x)$,*
2. *for all x, if $\nu(x) \leq k(x)$ then $\{\nu(x)\} = 0$ iff $\{\nu'(x)\} = 0$ and*
3. *for all x, y if $\nu(x) \leq k(x)$ and $\nu(y) \leq k(y)$ then $\{\nu(x)\} \leq \{\nu(y)\}$ iff $\{\nu(x)\} \leq \{\nu'(x)\} \leq \{\nu'(y)\}$*
4. *for all $\varphi \in \mathcal{G}$, $\nu \in \varphi$ iff $\nu' \in \varphi$.*

♢

Note that the equivalence relation is index by k and $\mathcal{G}$. Intuitively, if two valuations are bigger than the maximal constant k. The timed automaton can not distinguish between these valuations and they are considered as equivalent. The set of constraints $\mathcal{G}$ is used to ensure the soundness [32] of the abstract system constructed using the equivalence relation given above. By using the equivalence relation on valuations, a *normalization operator* on zones can be defined.

Definition 2.16 (Normalization Operator)**.** *Given a timed automaton $\mathcal{A}$, let $\mathcal{G}$ be a finite set of difference constraints and $k : \mathbb{X} \to \mathbb{Q}_0^+$ be a function mapping each clock x to the maximal constant $k(x)$ appearing in the guards or invariants in $\mathcal{A}$ containing x*

and $Z \in \Phi(\mathbb{X})$ be a zone. Then the semantics of the normalization operator on zones is defined as follows:

$$\mathsf{norm}_{k,\mathcal{G}}(Z) \stackrel{\mathsf{def}}{=} \{\nu \mid \nu \dot{\sim}_{k,\mathcal{G}} \nu', \nu' \in Z\}$$

$\diamondsuit$

Note that the equivalence relation is indexed by both a clock ceiling and a finite set of difference constraints, and so is the normalization operation. By using the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$ a finite transition system can be computed. For the rest of this thesis. We will refer to this transition system as *finite zone graph*. The finite zone graph uses the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$ to give a finite characterization of the transition relation $\rightarrow_{ZG}$ of the corresponding zone graph. The finite zone graph that we present is a maximal bound abstraction method. This is because the finite system preserves all the relevant information for values which are less than the maximal constant appearing in the automaton. In some cases, considering the maximal constant is not necessary and a coarser abstraction can take place [16]. There are several abstraction methods which consider different aspects. The abstraction we present is the most widely used and is implemented in tools such as UPPAAL [91] and KRONOS [140].

Definition 2.17 (Finite Zone Graph)**.** *The finite zone graph for timed automaton $\mathcal{A}$ is defined by the zone graph*

$$ZG_{k,\mathcal{G}}(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow_{k,\mathcal{G}}, c_0)$$

where:

- *$SConf(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ is the set of configurations consisting of pairs of a location $\ell \in L$ and a zone $Z \in \Phi(\mathbb{X})$.*
- *$c_0 = \langle \ell_0, \mathsf{norm}_{k,\mathcal{G}}(\{\nu_0\})\rangle$ is the initial configuration where $\nu_0(x) = 0$ for all clocks $x \in \mathbb{X}$.*
- *$\rightarrow_{k,\mathcal{G}} \subseteq SConf(\mathcal{A}) \times SConf(\mathcal{A})$ is the transition relation defined by the following rule:*

$$\frac{\langle \ell, Z\rangle \rightarrow_{ZG} \langle \ell', Z'\rangle}{\langle \ell, Z\rangle \rightarrow_{k,\mathcal{G}} \langle \ell', \mathsf{norm}_{k,\mathcal{G}}(Z')\rangle} \text{ if } Z = \mathsf{norm}_{k,\mathcal{G}}(Z)$$

$\diamondsuit$

Configurations in the finite zone graph are pairs consisting of locations and zones. The abstract transition relation is constructed by applying the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$ to the configurations in the concrete transition relation $\rightarrow_{ZG}$. The correctness of the abstraction is expressed in the following theorem. Soundness expresses that if a configuration is reached in the abstract system. Then, there is a corresponding reachable concrete configuration. Completeness expresses that if a configuration is reached in the concrete system. Then, there is a corresponding reachable abstract configuration.

$$
\begin{array}{lcl}
\mathsf{BF} & ::= & \mathcal{A}_i.\ell \mid \varphi, \\
\mathsf{CF} & ::= & \mathsf{BF} \mid \neg\mathsf{CF} \mid \mathsf{CF}_1 \wedge \mathsf{CF}_2, \\
\mathsf{EPF} & ::= & \exists\Diamond\mathsf{CF} \mid \exists\Box\mathsf{CF}, \\
\mathsf{APF} & ::= & \forall\Diamond\mathsf{CF} \mid \forall\Box\mathsf{CF} \mid \mathsf{CF}_1 \longrightarrow \mathsf{CF}_2, \\
\mathsf{PF} & ::= & \mathsf{EPF} \mid \mathsf{APF}.
\end{array}
$$

Table 2.1: Syntax of a subset of the Timed Computation Tree Logic.

Theorem 2.18. *Given timed automaton $\mathcal{A}$ with initial configuration $\langle \ell_0, \nu_0 \rangle, t_0$, maximal constant k and whose guards contain only a finite set of difference constraints denoted $\mathcal{G}$.*

1. *(Soundness) $\langle \ell_0, \{\nu_0\}\rangle(\rightarrow_{k,\mathcal{G}})^*\langle \ell_i, Z_i\rangle$ implies $\langle \ell_0, \{\nu_0\}\rangle, t_0(\xrightarrow{\lambda})^*\langle \ell_i, \nu_i\rangle, t_i$ for all $\nu_i \in Z_i$ such that $\nu_i(x) \leq k(x)$ for all $x \in \mathbb{X}$.*

2. *(Completeness) $\ell_0\{\nu_0\}, t_0(\xrightarrow{\lambda})^*\langle \ell_i, \nu_i\rangle, t_i$ with $\nu_i(x) \leq k(x)$ for all $x \in \mathbb{X}$ implies $\langle \ell_0, \{\nu_0\}\rangle(\rightarrow_{k,\mathcal{G}})^*\langle \ell_i, Z_i\rangle$ for some Z_i such that $\nu_i \in Z_i$.*

3. *(Finiteness) The transition relation $\rightarrow_{k,\mathcal{G}}$ is finite.*

For soundness, the existence of a simulation relation can be shown. Completeness follows immediately from the fact that the $\mathsf{norm}_{k,\mathcal{G}}$ operator applied to a zone produces an equal or bigger zone. The transition relation $\rightarrow_{k,\mathcal{G}}$ is finite because the set $\{\mathsf{norm}_{k,\mathcal{G}}(Z) \mid Z \in \Phi(\mathbb{X})\}$ is finite.

2.2 Timed Computation Tree Logic (TCTL)

Timed Computation Tree Logic(TCTL for short) is a real-time variant of *Computation Tree Logic*(CTL) conceived to express properties of timed automata. TCTL enables to express properties such as; a given stated has to reached within j time units. Timed CTL is sufficiently expressive to allow the formulation of an important set of real-time system properties. In the following, we present the logic of UPPAAL which is a subset of the Timed Computation Tree Logic. Our presentation follows [10, 105].

2.2.1 Syntax

Informally, this logic allows us to express that the following properties φ on configurations should hold along the computation path of a given transition system.

- $\exists\Diamond\varphi$ expresses that there exists a computation path along which eventually φ holds.
- $\exists\Box\varphi$ expresses that there exists a computation path along which φ always holds.
- $\forall\Diamond\varphi$ express that along all computation paths φ eventually holds.
- $\forall\Box\varphi$ expresses that along all computation paths φ always holds.
- $\varphi_1 \longrightarrow \varphi_2$ expresses that each occurrence of φ_1 eventually leads to an occurrence of φ_2.

For the rest of this section let us fix an automaton $\mathcal{N} = \mathcal{A}_1 \| \dots \| \mathcal{A}_n$ with transition system $\mathcal{TS}(\mathcal{N})$ set of clocks $\mathbb{X}$ and initial configuration C_0. The formal definition of TCTL is given by the grammar in Table 2.1. *Basic Formulae* BF, these formulae are of the form $\mathcal{A}_i.\ell$ or φ. The formula $\mathcal{A}_i.\ell$ asserts that the current location of the i-th time automaton has to be ℓ. The formula φ is a constraint over $\mathbb{X}$. *Configuration Formulae* CF, these formulae express constraints on the valuations of a configuration. Configuration formulae can be a basic formula BF, a negation of a configuration formula or a conjunction of configuration formulas. *Path Formulae* PF, by starting at a configuration, these formulae establishes constraints on the future behavior of the system. Path formulae are divided into existential and universal path formulae. *Existential Path Formulae* EPF and *Universal Path Formulae* APF enable to express properties of the future behavior of the system on a quantified manner. The intuition of quantified path formulas e.g. $\exists\Diamond\mathsf{CF}$ has been given above.

Example 2.19. Consider the timed automaton in Figure 2.2. The property. For all the possible computations, clocks x and y are always equal or one of the clocks is equal to zero. Can be expressed by the following $\forall\Box\ x = y \vee x = 0 \vee y = 0$. Note that our grammar do not consider disjunctions. However, disjunctions can be represented using conjunctions and negations [10]. The formula given above can be expressed as $\forall\Box\ \neg(x \neq y \wedge x \neq 0 \wedge y \neq 0)$. ♢

2.2.2 Semantics

The semantics of TCTL formulae is defined for configurations of the form $\langle \ell, \nu \rangle, t$. Configuration formulae can be evaluated using the valuations. Path formulae are evaluated over all divergent time paths. The following notation will come useful in defining the semantics of TCTL.

Definition 2.20 (Additional Notation)**.** *Let $\pi \in \mathcal{TS}(\mathcal{N})$ i.e. π is a computation path of $\mathcal{N}$ starting at some $\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \in C_0$ of the form:*

$$\pi = \langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \xrightarrow{\lambda_0} \langle \overrightarrow{\ell_1}, \nu_1 \rangle, t_1 \xrightarrow{\lambda_1} \langle \overrightarrow{\ell_2}, \nu_2 \rangle, t_2 \xrightarrow{\lambda_2} \langle \overrightarrow{\ell_3}, \nu_3 \rangle, t_3 \dots$$

and let $t \in \mathbb{R}_0^+$. *We denote by* $\pi(t)$ *the set of configurations in* π *at time* t, *Formally:*

$$\pi(t) \stackrel{\mathsf{def}}{=} \{\langle \overrightarrow{\ell}, \nu\rangle \mid \exists i \in \mathbb{N}_0 \bullet (t_i \leq t \leq t_{i+1} \wedge \overrightarrow{\ell} = \overrightarrow{\ell_i} \wedge \nu = \nu_i + t - t_i)\}$$

Finally, let $\langle \overrightarrow{\ell}, \nu\rangle \in \mathit{Conf}(\mathcal{N})$ *denote a configuration* $\langle \overrightarrow{\ell}, \nu\rangle, t' \in \mathit{Conf}(\mathcal{N})$ *for some* $t' \in \mathbb{R}_0^+$. ♢

We now introduce a binary satisfaction relation $\models$ between time stamped configurations $\langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0$ and formulas F of the UPPAAL logic described in Table 2.1. Written as

$$\langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0 \models F.$$

The formal definition of the satisfaction relation is given in Table 2.2. The definition is an inductive definition. Given a configuration. If the formula is a configuration formula, then the formula can be directly evaluated in the given configuration. If the formula is a basic formula, then the interpretations of conjunctions and negations are as usual using the given configuration. For path formulas. In the case of existential path formulae, the formula holds if and only if there exist some path starting at $\langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0$ which models the corresponding configuration formula. In the case of universal path formulae, the formula holds if and only if all paths starting at $\langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0$ model the corresponding configuration formula.

The satisfaction relation is defined on configurations. We now lift the satisfaction relation $\models$ to transition systems as follows:

$$\begin{array}{lll} \mathcal{TS}(\mathcal{N}) \models \mathsf{EPF} & \text{iff} & \langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0 \models \mathsf{EPF} \text{ for some } \langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0 \in C_0, \\ \mathcal{TS}(\mathcal{N}) \models \mathsf{APF} & \text{iff} & \langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0 \models \mathsf{APF} \text{ for all } \langle\overrightarrow{\ell_0}, \nu_0\rangle, t_0 \in C_0. \end{array}$$

Note that by definition of transition system, C_0 can have only one element or none. Therefore, if C_0 is empty an EPF formula is never satisfied whereas any APF formula is trivially satisfied.

$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \mathcal{A}_i.\ell$	iff	$\ell_{0,i} = \ell$, i.e. the ith component of $\overrightarrow{\ell_0}$ is ℓ,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \varphi$	iff	$\nu_0 \models \varphi$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \neg \mathsf{CF}$	iff	$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \not\models \mathsf{CF}$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \mathsf{CF}_1 \wedge \mathsf{CF}_2$	iff	$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \mathsf{CF}_1$ and $\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \mathsf{CF}_2$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \exists\Diamond \mathsf{CF}$	iff	$\exists \pi \in \mathcal{TS}(\mathcal{N}), t \in \mathbb{R}_0^+, \langle \overrightarrow{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet$ $t_0 \leq t \wedge \langle \overrightarrow{\ell}, \nu \rangle \in \pi(t) \wedge \langle \overrightarrow{\ell}, \nu \rangle, t \models CF$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \forall\Box \mathsf{CF}$	iff	$\forall \pi \in \mathcal{TS}(\mathcal{N}), t \in \mathbb{R}_0^+, \langle \overrightarrow{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet$ $t_0 \leq t \wedge \langle \overrightarrow{\ell}, \nu \rangle \in \pi(t) \implies \langle \overrightarrow{\ell}, \nu \rangle, t \models CF$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \exists\Box \mathsf{CF}$	iff	$\exists \pi \in \mathcal{TS}(\mathcal{N}) \bullet \forall t \in \mathbb{R}_0^+, \langle \overrightarrow{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet$ $t_0 \leq t \wedge \langle \overrightarrow{\ell}, \nu \rangle \in \pi(t) \implies \langle \overrightarrow{\ell}, \nu \rangle, t \models CF$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \forall\Box \mathsf{CF}$	iff	$\forall \pi \in \mathcal{TS}(\mathcal{N}) \bullet \exists t \in \mathbb{R}_0^+, \langle \overrightarrow{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet$ $t_0 \leq t \wedge \langle \overrightarrow{\ell}, \nu \rangle \in \pi(t) \wedge \langle \overrightarrow{\ell}, \nu \rangle, t \models CF$,
$\langle \overrightarrow{\ell_0}, \nu_0 \rangle, t_0 \models \mathsf{CF}_1 \longrightarrow \mathsf{CF}_2$	iff	$\forall \pi \in \mathcal{TS}(\mathcal{N}) \bullet \forall t \in \mathbb{R}_0^+, \langle \overrightarrow{\ell}, \nu \rangle \in \mathit{Conf}(\mathcal{N}) \bullet$ $t_0 \leq t \wedge \langle \overrightarrow{\ell}, \nu \rangle \in \pi(t) \wedge \langle \overrightarrow{\ell}, \nu \rangle, t \models CF_1$ implies $\langle \overrightarrow{\ell}, \nu \rangle, t \models \forall\Diamond CF_2$.

Table 2.2: Semantics of Timed Computation Tree Logic

Chapter 3

Semantic Optimizations for TDMA Systems

Contents

In this chapter we study key principles of TDMA based systems and then use the theory of timed automata to formalize these principles. Later we exploit these principles for reducing the time complexity of the verification task.

In TDMA based systems the information is transmitted in frames or cycles. Cycles have a fix length. Once a cycle ends, a new cycle starts. This behavior of the system leads to our definition of *periodic cyclic timed automata*. As TDMA systems are based on cycles. This class of automata has the property that a given location has to be infinitely often visited. This property is similar to the acceptance condition of *timed Büchi automata* [5]. The most important difference is that a given configuration has to be visited at regular points of time. These points of time are multiples of a natural number and a fix number, which we call the *period*.

The structure of TDMA based systems ensures the absence of collisions in a shared medium. This fact suggest an independence among the TDMA components. This observation leads to our definition of *disjoint activity*. Under *activity* of a timed automaton, we refer to the points in time where the automaton can perform discrete transitions. If the activity of two timed automata do not overlap, we say that the automata have disjoint activity.

The components of the TDMA system, share their information in their corresponding time slots in consecutive succession. This observation and the notion of disjoint activity,

lead to the definition of *sequentialisation*. If two timed automata are sequentialisable, then their activity is mostly disjoint and one automaton executes a number of actions before the other one in every TDMA cycle. The independence of actions and the succession of actions from different automata, propose the use of a more specific operator than the parallel composition operator. We therefore present a *concatenation operator* on periodic cyclic timed automata with disjoint activity.

For sequentialisable periodic cyclic timed automata. We study the relationship between the systems composed by using the concatenation operator and systems composed using the parallel composition operator. Our results shows that the resulting systems are bisimilar. In addition, we show that the use of the concatenation operator lead to quadratic speed ups on the verification time complexity. The reduction on the verification time complexity, is due to a reduced number of edges on the system obtained using the concatenation operator.

Contributions

The key technical contributions that are described in this chapter are summarized as follows:

- We introduce the class of periodic cyclic timed automata. We then show that this class of automata is closed under application of the parallel composition operator.
- For the theory of timed automata. We introduce the notions of activity, disjoint activity and sequentialisability.
- We introduce a concatenation operator on sequentialisable periodic cyclic timed automata. We show that the class of periodic cyclic timed automata is closed under the application of the concatenation operator.
- For sequentialisable periodic cyclic timed automata. We show that the resulting system obtained using the concatenation operator is bisimilar to the corresponding one obtained using the parallel product operator. We then introduce the notion of enable edges and show that using the concatenation operator on sequentialisable timed automata yield quadratic speed ups.

3.1 Periodic Cyclic Timed Automata

Timed automata models of, e.g. TDMA-based protocols can be cyclic and periodic in the following sense. Intuitively, a timed automaton is cyclic if the initial location is visited infinitely often on all computations, the corresponding configurations are called start configuration. A timed automaton is periodic with period pt if configurations containing the initial location are reached only at integer multiples of the period and are reached from a unique final location. In the following, we formally define periodic cyclic timed automata in terms of the new notions of start, restart, and final configurations (cf. Figure 3.1).

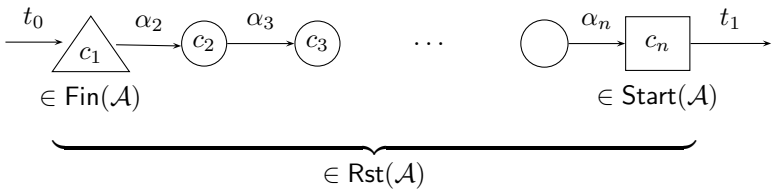


Figure 3.1: Start configurations are in the initial location and have an action predecessor and a delay successor, final configurations a delay predecessor and an action successor. Configurations on an action-only path between a final and a start configuration are called restart configurations.

Definition 3.1 (Start, Restart and Final Configurations)**.** *Let* $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *be a timed automaton. The set* $\mathsf{Start}(\mathcal{A})$ *of* start configurations $\mathcal{A}$ *consists of those configurations of* $\mathcal{TS}(\mathcal{A})$ *that are at location* ℓ_0 *and occur in a computation* $\pi \in \mathcal{TS}(\mathcal{A})$ *as source of a delay transition and as destination of an action transition, i.e.*

$$\mathsf{Start}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \{c = \langle \ell_0, \nu \rangle, t \in \mathit{Conf}(\mathcal{A}) \mid \exists \pi \in \mathcal{TS}(\mathcal{A}), m \in \mathbb{N}_0 \bullet \\ \pi_m = c \wedge \lambda_m \in \mathbb{R} \wedge (\lambda_{m-1} \in \Sigma \vee m = 0)\}.$$

The set $\mathsf{Rst}(\mathcal{A})$ *of* restart configurations *consists of those configuration of* $\mathcal{TS}(\mathcal{A})$ *that occur in a computation* $\pi \in \mathcal{TS}(\mathcal{A})$ *as action-predecessor of a start configuration, i.e.*

$$\mathsf{Rst}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \{c \in \mathit{Conf}(\mathcal{A}) \mid \exists \pi \in \mathcal{TS}(\mathcal{A}), m, i \in \mathbb{N}_0 \bullet m \leq i \wedge \pi_m = c \\ \wedge \pi_i \in \mathsf{Start}(\mathcal{A}) \wedge \pi_m \xrightarrow{\lambda_m} \ldots \xrightarrow{\lambda_{i-1}} \pi_i \wedge \forall m \leq j \leq i \bullet \lambda_j \in \Sigma\}.$$

The set $\mathsf{Fin}(\mathcal{A})$ *of* final configurations *consists of the maximal restart configurations of* $\mathcal{TS}(\mathcal{A})$*, that is, restart configurations which are the destination of a delay transition, i.e.*

$$\mathsf{Fin}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \{c \in \mathsf{Rst}(\mathcal{A}) \mid \exists \pi \in \mathcal{TS}(\mathcal{A}), m \in \mathbb{N}_0 \bullet \\ \pi_m = c \wedge (\lambda_{m-1} \in \mathbb{R} \vee m = 0)\}.$$

The set L_{rst} *of* restart locations *consists of those locations that occur in a restart configuration, i.e.*

$$L_{\mathsf{rst}} \stackrel{\mathsf{def}}{=} \{\ell \in L \mid \exists \nu : \mathbb{X} \to \mathbb{R}, t \in \mathbb{R} \bullet \langle \ell, \nu \rangle, t \in \mathsf{Rst}(\mathcal{A})\}.$$

$\diamondsuit$

Figure 3.1, depicts a fragment of a computation path. In the computation path the configuration c_1 is a final configuration, because it is the result of a delay transition and the successor is due to an action transition. The configuration c_n is a start configuration, because it is at the initial location, the source of an action transition and its successor is due to a delay transition. The set of restart configurations is the set consisting of all the configurations reaching c_n via action transitions. The set of restart locations, consist of all the locations in the configurations of the set of restart configurations. We illustrate the set of restart locations with an example.

Example 3.2. First, consider the timed automaton $\mathcal{A}_1$ from Figure 3.2. The timed automaton corresponds to a sensor of the fire alarm system presented in Section 1.2. At point of time 1500 there is a transition from configuration $\langle \ell_3, \nu(x_1) = 1500 \rangle, 1500$ leading to the configuration $\langle \ell_0, \nu(x_1) = 0 \rangle, 1500$. The configuration $\langle \ell_0, \nu(x_1) = 0 \rangle, 1500$ is a start configuration, because it is at location ℓ_0, its possible successors are delay successors, and it is reached via an action transition. The configuration $\langle \ell_3, \nu(x_1) = 1500 \rangle, 1500$ is a final configuration, because it has no delay predecessors and reaches a start configuration via action transitions. If we analyze the automaton carefully, we realize that the set of restart locations consist of locations ℓ_0 and ℓ_3.

Now, consider Figure 3.3. Corresponding to the parallel product of the sensors 1 and 2 from the fire alarm system presented in Section 1.2. Consider a computation

$$
\begin{aligned}
&\ldots \xrightarrow{\lambda_i}{}^{*} \langle (\ell_3, \ell_3), \nu(x) = \nu(y) = 1500 \rangle, 1500 \\
&\xrightarrow{\lambda_{i+1}}{}^{*} \langle (\ell_0, \ell_3), \nu(x) = 0, \nu(y) = 1500 \rangle, 1500 \\
&\xrightarrow{\lambda_{i+2}}{}^{*} \langle (\ell_0, \ell_0), \nu(x) = 0, \nu(y) = 0 \rangle, 1500 \\
&\xrightarrow{\lambda_{i+3}}{}^{*} \ldots
\end{aligned}
$$

Then, configuration $\langle \ell_3, \ell_3, \nu(x) = \nu(y) = 1500 \rangle, 1500$ is a final configuration. Configuration $\langle \ell_0, \ell_0, \nu(x) = 0, \nu(y) = 0 \rangle, 1500$ is a start configuration and configuration $\langle \ell_0, \ell_3, \nu(x) = 0, \nu(y) = 1500 \rangle, 1500$ is in the set of restart configurations. The set of restart locations consist of the locations in the configurations of the above mentioned configurations and the location (ℓ_0, ℓ_3). ♢

By using the definitions of restart, start and final configurations we are ready to define the class of periodic cyclic timed automata. Intuitively, a timed automaton is a periodic cyclic timed automaton. If all its computations visit a start configuration infinitely often. An additional constraint is that the start configurations can be only visited at determined points of time. These points of time are multiples of the period and the natural numbers. Another constraint, is that the automaton has only one final location. i.e. all the final configurations are at location ℓ_{fin}. Figure 3.2 exhibit a periodic cyclic

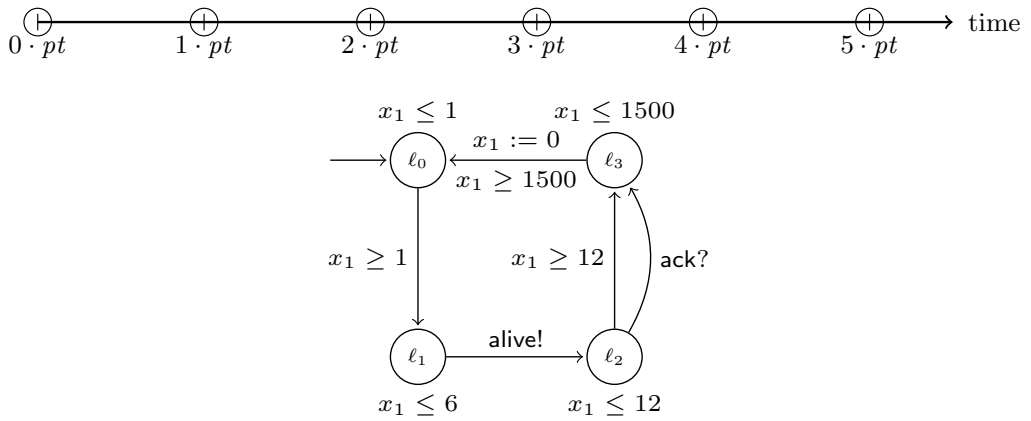


Figure 3.2: Period points of time of a sensor. Bottom, timed automaton corresponding to sensor 1 of the fire alarm system presented in Section 1.2. Top, period points of sensor 1, the points of time correspond to the integer multiples of the TDMA cycle, with period $pt = 1500$.

timed automaton. The automaton corresponds to a sensor of the fire alarm system described in Section 1.2. The time axis depicts the points of time at which a start configuration is visited. Note that the final location of the automaton is ℓ_3. The formal definition of periodic cyclic timed automata follows.

Definition 3.3 (Periodic Cyclic)**.** *A timed automaton* $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *is called* periodic cyclic *with period* $pt \in \mathbb{R}$ *if and only if each computation comprises infinitely many start configurations which occur at a regular period of time and if there is a unique* final location ℓ_{fin}*, i.e.*

$$
\begin{aligned}
\mathsf{pecy}(\mathcal{A}, pt) \overset{\mathsf{def}}{\iff} \forall \pi \in \mathcal{TS}(\mathcal{A}), p \in \mathbb{N}_0 \; \exists c = (\langle \ell, \nu \rangle, t) \in \mathsf{Start}(\mathcal{A}), i \in \mathbb{N}_0 \bullet \\
\quad \pi_i = c \wedge t = pt \cdot p \; \wedge \\
\forall \pi \in \mathcal{TS}(\mathcal{A}), c = (\langle \ell, \nu \rangle, t) \in \mathsf{Start}(\mathcal{A}), i \in \mathbb{N}_0 \; \bullet \\
\quad \pi_i = c \Rightarrow \exists p \in \mathbb{N}_0 \bullet t = pt \cdot p \; \wedge \\
\exists \ell_{\mathsf{fin}} \in L \; \forall (\langle \ell, \nu \rangle, t) \in \mathsf{Fin}(\mathcal{A}) \bullet \ell = \ell_{\mathsf{fin}}.
\end{aligned}
$$

♢

Now we will show that the class of periodic cyclic timed automata is closed under the application of the parallel product operator. This is important, because it ensures that the properties of periodic cyclic timed automata are preserved after applying the parallel

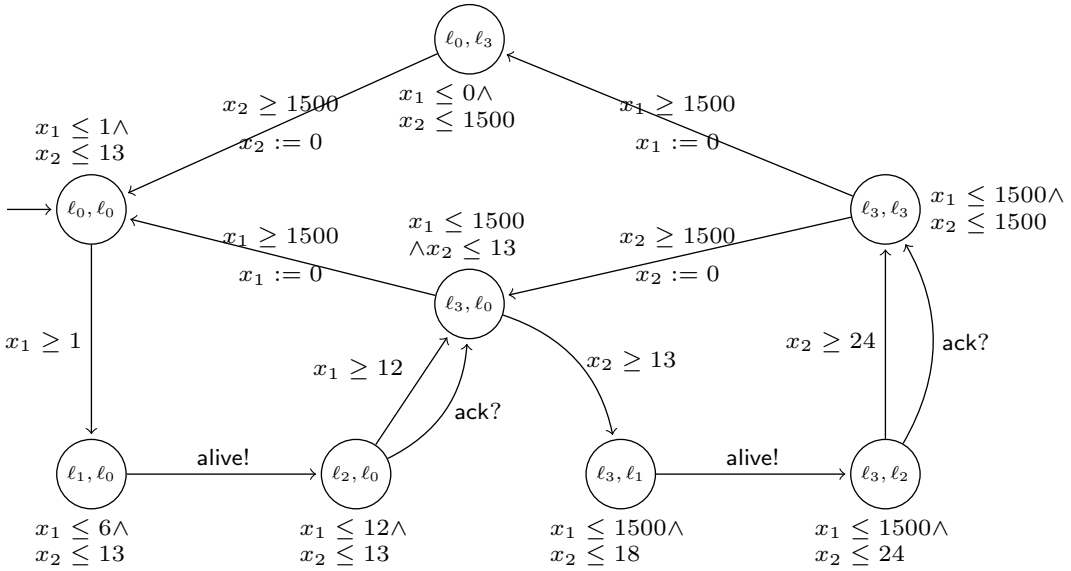


Figure 3.3: Reachable fragment of the timed automaton obtained by applying the parallel composition operator on sensor 1 and sensor 2 form the fire alarm system presented in Section 1.2.

product operator. In the following, we define a projection on computations. Given an automaton which is the result of the application of the parallel product operator on two timed automata. The projection function applied on a computation of the given timed automaton will project it onto a computation of one of its components. This definition will come useful for the proofs in this chapter.

Definition 3.4 (Projection)**.** *In the following, we assume that in computations of $\mathcal{A}_1 \| \mathcal{A}_2$, each transition is implicitly associated with the information which edges have been considered to justify the transition. That is, we assume that for each λ_j, $j \in \mathbb{N}$, occurring in a computation $\pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ there is a pair which $(\varepsilon_1(\lambda_j), \varepsilon_2(\lambda_j)) \in 2^{E_1} \times 2^{E_2}$ gives the involved edges. For asynchronous edges, the component corresponding to the timed automaton which* does not *take a transition, is the empty set $\emptyset$. If λ_j is a delay transition, then $\varepsilon_1(\lambda_j) = \varepsilon_2(\lambda_j) = \emptyset$.*

Let

$$\begin{aligned}\pi = \langle(\ell_{1,0}, \ell_{2,0}), \nu_0\rangle, t_0 &\xrightarrow{\lambda_{0,1}} \ldots \xrightarrow{\lambda_{0,n_0}} \langle(\ell_{1,0,n_0}, \ell_{2,0,n_0}), \nu_{0,n_0}\rangle, t_{0,n_0} \\ &\xrightarrow{\lambda_0} \langle(\ell_{1,1}, \ell_{2,1}), \nu_1\rangle, t_1 \xrightarrow{\lambda_{1,1}} \ldots \xrightarrow{\lambda_{1,n_1}} \langle(\ell_{1,1,n_1}, \ell_{2,1,n_1}), \nu_{1,n_1}\rangle, t_{1,n_1} \\ &\xrightarrow{\lambda_1} \ldots\end{aligned}$$

be a computation of the parallel composition of $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *which is renumbered such that* $\lambda_{j,1}, \ldots, \lambda_{j,n_j}$, $j \in \mathbb{N}_0$, *are maximal sequences of actions of* $\mathcal{A}_2$, *i.e.* $\varepsilon_1(\lambda_{j,i}) = \emptyset$ *and* $\varepsilon_2(\lambda_{j,i}) \neq \emptyset$ *for all* $j \in \mathbb{N}_0$ *and* $1 \leq i \leq n_j$.

Then the projection *of* π onto $\mathcal{A}_1$, *denoted by* $\Gamma_1(\pi)$, *is defined as*

$$\Gamma_1(\pi) \stackrel{\mathsf{def}}{=} \langle(\ell_{1,0}), \nu_0|_{\mathbb{X}_1}\rangle, t_0 \xrightarrow{\tilde{\lambda}_0} \langle(\ell_{1,1}), \nu_1|_{\mathbb{X}_1}\rangle, t_1 \xrightarrow{\tilde{\lambda}_1} \ldots$$

where $\nu_i|_{\mathbb{X}_1}$ *denotes the standard function restriction of valuation* ν_i *to the range* $\mathbb{X}_1$, $i \in \mathbb{N}_0$, *and where* $\tilde{\lambda}_i$ *denotes action* α *if* $\varepsilon_1(\lambda_i) = \{(\ell, \alpha, \varphi, Y, \ell')\}$, *and just* λ_i *otherwise. The projection of* π *onto* $\mathcal{A}_2$, *denoted by* $\Gamma_2(\pi)$, *is defined analogously.* ♢

For a timed automaton which is the result of applying the parallel product on two timed automata. Using the projection defined above. A computation of the timed automaton will be projected to a computation on one of its components. Given a computation, the following lemma states that the computation obtained by projecting it onto a component is indeed a computation of the component.

Lemma 3.5. *Let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be timed automata. Then the projection of a computation of* $\mathcal{A}_1 \| \mathcal{A}_2$ *onto* $\mathcal{A}_i$ *is a computation of* $\mathcal{A}_i$, $i = 1, 2$, *i.e.*

$$\forall \pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2), i \in \{1, 2\} \bullet \Gamma_i(\pi) \in \mathcal{TS}(\mathcal{A}_i).$$

Proof. We consider the case for $\mathcal{A}_1$ the proof for $\mathcal{A}_2$ can be done analogously. We proceed by induction on the index i of the computation π. We show that for every i, there exist a j s.t. $\Gamma_1(\pi)_j$ is reachable in $\mathcal{TS}(\mathcal{A}_1)$ and if $\varepsilon_1(\lambda_i) \neq \emptyset$ or $\lambda_i \in \mathbb{R}$ then $\pi_i = \langle(\ell_{1,i}, \ell_{2,i}), \nu_i\rangle, t_i$ and $\Gamma_1(\pi)_j = \langle(\ell_{1,i}), \nu_i|_{\mathbb{X}_1}\rangle, t_i$.
Base case: $i = j = 0$, then $(\Gamma_1(\pi))_0 = \langle\ell_0, \nu_0\rangle, t_0 = c_0$.
Induction step: by IH $(\Gamma_1(\pi))_j$ is reachable, we consider two cases:

- $\varepsilon_1(\lambda_i) \neq \emptyset$ or $\lambda_i \in \mathbb{R}$ i.e. a transition involving $\mathcal{A}_1$ with $\pi_i = \langle(\ell_{1,i}, \ell_{2,i}), \nu_i\rangle$, t_i and $(\Gamma_1(\pi))_j = \langle(\ell_{1,i}), \nu_i|_{\mathbb{X}_1}\rangle, t_i$. With

  $$\langle(\ell_{1,i}, \ell_{2,i}), \nu_i\rangle, t_i \xrightarrow{\lambda_i} \langle(\ell_{1,i+1}, \ell_{2,i+1}), \nu_{i+1}\rangle, t_{i+1}$$

  We have two cases:

  - $\lambda_i \in \mathbb{R}$. There is a time transition and $\nu_{i+1} \models I_1(\ell_{1,i+1}) \wedge I_2(\ell_{2,i+1})$. By def. of rewriting and Γ we have $\Gamma_1(\pi)_j \xrightarrow{\lambda_i} \Gamma_1(\pi)_{j+1}$ with $\Gamma_1(\pi)_{j+1} = \langle\ell_{1,i+1}b, \nu_{i+1}|_{\mathbb{X}_1}\rangle, t_{i+1}$. Then $\nu_{i+1}|_{\mathbb{X}_1} = \nu_j$, $\ell_{1,i+1} = \ell_{1,j}$ and $\nu_j \models I(\ell_{1,j})$. Therefore, $\Gamma_1(\pi)_{j+1}$ is reachable.

– $\varepsilon_1(\lambda_i) \neq \emptyset$. There is an local transition or a synchronization transition in $\mathcal{A}_1$. By definition of the projection we have $\varepsilon_1(\lambda_i) = \{e = (\ell_{1,i}, \lambda_i, \varphi, Y, \ell_{1,i+1})\}$, s.t. $\nu_i \models \varphi$, $\nu_{i+1} = \nu[Y := 0]$ and $\nu_{i+1} \models I_1(\ell_{1,i+1}) \wedge I_2(\ell_{2,i+1})$. By IH $\Gamma_1(\pi)_j = \langle(\ell_{1,i}), \nu_i|_{\mathbb{X}_1}\rangle, t_i$ for some j and by def of Γ it follows, $\Gamma_1(\pi)_{j+1} = \langle\ell_{1,i+1}, \nu_{i+1}|_{\mathbb{X}_1}\rangle, t_{i+1}$. By edge e and the fact that $\nu_{i+1}|_{\mathbb{X}_1} \models I(\ell_{1,i+1})$. $\Gamma_1(\pi)_{j+1}$ is reachable from $\Gamma_1(\pi)_j$.

- $\lambda_i \in \Sigma_2$. Then the computation fragment starting at π_i where λ_i is being renamed accordingly to the definition is of the form:

$$\pi_i \xrightarrow{\lambda_{i,0}} \pi_{i+1} \xrightarrow{\lambda_{i,1}} \pi_{i+2} \xrightarrow{\lambda_{i,2}} \ldots \xrightarrow{\lambda_{i,k}} \pi_{i+k} \xrightarrow{\lambda_{i+1}} \pi_{i+k+1}$$

  for some $k \in \mathbb{N}$ and $k > 0$. and for all $k' \in \mathbb{N}$ such that $0 < k' \leq k$, $\lambda_{k'} \in \Sigma_2$. Therefore, $\pi_{i+k} = \langle\ell_{i+k}, \nu_i|_{\mathbb{X}_1} \cup \nu_{i+k}|_{\mathbb{X}_2}\rangle, t_i$ with $\ell_{i+k} = (\ell_{1,i}, \ell_{2,i,k})$ and $\varepsilon_1(\lambda_{i+k}) \neq \emptyset$ or $\lambda_{i+k} \in \mathbb{R}$. In particular, we are in the above case involving an action transition of $\mathcal{A}_1$ or a time delay. This have been shown above.

Therefore, $\Gamma_1(\pi) \in \mathcal{TS}(\mathcal{A}_1)$ and $\Gamma_2(\pi) \in \mathcal{TS}(\mathcal{A}_2)$. □

Let us illustrate with an example the application of the projection function Γ to a computation.

Example 3.6. Consider Figure 3.3. The automaton is the parallel product of sensors 1 and 2 from the fire alarm system presented in Section 1.2. We refer to sensor 1 and 2 by $\mathcal{A}_1$ and $\mathcal{A}_2$ respectively. Consider a computation $\pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ such that,

$$\begin{aligned}\pi = \quad & \ldots \xrightarrow{0}{}^{*} \langle(\ell_3, \ell_3), \nu(x) = \nu(y) = 1500\rangle, 1500 \\ & \xrightarrow{0}{}^{*} \langle(\ell_0, \ell_3), \nu(x) = 0, \nu(y) = 1500\rangle, 1500 \\ & \xrightarrow{0}{}^{*} \langle(\ell_0, \ell_0), \nu(x) = 0, \nu(y) = 0\rangle, 1500 \\ & \xrightarrow{\lambda_i}{}^{*} \ldots\end{aligned}$$

Then, the projection of π onto sensor 1 is

$$\begin{aligned}\Gamma_1(\pi) = \quad & \ldots \xrightarrow{0}{}^{*} \langle\ell_3, \nu(x) = 1500\rangle, 1500 \\ & \xrightarrow{0}{}^{*} \langle\ell_0, \nu(x) = 0\rangle, 1500 \\ & \xrightarrow{\lambda_j}{}^{*} \ldots\end{aligned}$$

Because of Lemma 3.5, the resulting projection is a computation of sensor 1, i.e. $\Gamma_1(\pi) \in \mathcal{TS}(\mathcal{A}_1)$. ♢

The following theorem states that the class of periodic cyclic timed automata is closed under application of the parallel composition operator. This theorem will come useful in the next section for showing properties of sequentialisable timed automata.

Theorem 3.7. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be periodic cyclic timed automata with period $pt \in \mathbb{R}$. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is periodic cyclic with period pt.*

Proof. By definition of periodic cyclic we have

$$
\begin{aligned}
&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \Rightarrow \\
&\forall \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), p \in \mathbb{N}_0\ \exists c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2), i \in \mathbb{N}_0 \bullet \\
&\quad \pi_i = c \wedge pt \cdot p = t \wedge \\
&\forall \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)\ , i \in \mathbb{N}_0 \bullet \\
&\quad \pi_i = c \Rightarrow \exists p \bullet t = pt \cdot p\ \wedge \\
&\exists \ell_{\mathsf{fin}} \in L\ \forall (\langle \ell, \nu\rangle, t) \in \mathsf{Fin}(\mathcal{A}_1||\mathcal{A}_2) \bullet \ell = \ell_{\mathsf{fin}}
\end{aligned}
$$

We split our proof in three and show first :

$$
\begin{aligned}
&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \Rightarrow \\
&\forall \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), p \in \mathbb{N}_0\ \exists c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2), i \in \mathbb{N}_0 \bullet \\
&\quad \pi_i = c \wedge pt \cdot p = t
\end{aligned}
$$

We negate formula and show that it is unsatisfiable.

$$
\begin{aligned}
&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \wedge \\
&\exists \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), p \in \mathbb{N}_0\ \forall c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2), i \in \mathbb{N}_0 \bullet \\
&\quad \pi_i = c \Rightarrow pt \cdot p \neq t
\end{aligned}
$$

The formula states that there exist a computation π for which at period p there is no start configuration. That is all the start configurations in π do not occur at period p. Assume, that such a sequence π and period p exists. We show, that this yields a contradiction. By Lemma 3.5, $\Gamma_1(\pi) \in \mathcal{TS}(\mathcal{A}_1)$ and $\Gamma_2(\pi) \in \mathcal{TS}(\mathcal{A}_2)$. Since $\mathsf{pecy}(\mathcal{A}_1, pt)$ and $\mathsf{pecy}(\mathcal{A}_2, pt)$. Computation $\Gamma_1(\pi)$ has at point time $pt \cdot p$ a transition $\langle \ell_1, \nu_1\rangle, pt \cdot p \xrightarrow{\alpha} \langle \ell_{ini_1}, \nu_1'\rangle, pt \cdot p$ for some edge $(\ell_1, \alpha, \varphi, Y, \ell_{ini_1}) \in E_1$ with $\nu_1 \models \varphi$, $\nu_1' = \nu_1[Y := 0]$ and $\nu_1' \models I_1(\ell_1')$. Analogously for $\mathcal{A}_2$. By the semantics of local transitions in networks we have that there is a local transition corresponding to $\mathcal{A}_1$ $\langle(\ell_1, \ell_2, \nu_1 \cup \nu_2\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_{ini_1}, \ell_2), \nu_1' \cup \nu_2\rangle, pt \cdot p$ and a local transition corresponding to $\mathcal{A}_2$ $\langle(\ell_1, \ell_2, \nu_1 \cup \nu_2\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_1', \ell_{ini_2}), \nu_1 \cup \nu_2'\rangle, pt \cdot p$ in particular $\mathcal{A}_1$, $\mathcal{A}_2$ remain at ℓ_{ini_1}, ℓ_{ini_2} respectively at time point $pt \cdot p$. Therefore, we have

$$\ldots \xrightarrow{\alpha} \langle(\ell_1, \ell_2, \nu\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_{ini_1}, \ell_2), \nu'\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_{ini_1}, \ell_{ini_2}), \nu''\rangle, pt \cdot p$$

or

$$\ldots \xrightarrow{\alpha} \langle(\ell_1, \ell_2, \nu\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_1, \ell_{ini_2}), \nu'''\rangle, pt \cdot p \xrightarrow{\alpha} \langle((\ell_{ini_1}, \ell_{ini_2}), \nu''\rangle, pt \cdot p$$

therefore there exist a $i \in \mathbb{N}_0$ such that $\pi_i = \langle((\ell_{ini_1}, \ell_{ini_2}), \nu_1' \cup \nu_2\rangle, pt \cdot p$ and $\pi_i \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)$ this yields a contradiction. Therefore, $\pi \notin \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2)$.
We now show,

$$\begin{aligned}&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \Rightarrow \\ &\forall \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)\ , i \in \mathbb{N}_0 \bullet \\ &\quad \pi_i = c \Rightarrow \exists p \bullet t = pt \cdot p\end{aligned}$$

We negate the formula and show that it is unsatisfiable,

$$\begin{aligned}&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \Rightarrow \\ &\exists \pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2), c = (\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)\ , i \in \mathbb{N}_0 \bullet \\ &\quad \pi_i = c \wedge \forall p \bullet t \neq pt \cdot p\end{aligned}$$

The formula states that there is a computation in which there is a start configuration which has a time stamp which is not a multiple of pt for all $p \in \mathbb{N}_0$. By definition of $\mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)$ we have $\pi_i = \langle \ell, \nu\rangle, t$ with $\ell = (\ell_{0_1}, \ell_{0_2})$. By Lemma 3.5 $\Gamma_1(\pi) \in \mathcal{TS}(\mathcal{A}_1)$ and there exist $j \in \mathbb{N}$ s.t. $(\Gamma_1(\pi))_j = \langle \ell_{0_1}, \nu|_{\mathbb{X}_1}\rangle, t$. Therefore, $(\Gamma_1(\pi))_j \in \mathsf{Start}(\mathcal{A}_1)$. By assumption, $\mathcal{A}_1$ is periodic cyclic with period pt. Thus, $(\Gamma_1(\pi))_j$ with time stamp t is such that $t = pt \cdot p'$ for some $p' \in \mathbb{N}$, which is a contradiction.
Finally, we show:

$$\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \Rightarrow \exists \ell_{\mathsf{fin}} \in L\ \forall(\langle \ell, \nu\rangle, t) \in \mathsf{Fin}(\mathcal{A}_1||\mathcal{A}_2) \bullet \ell = \ell_{\mathsf{fin}}$$

We show that the negation is unsatisfiable

$$\begin{aligned}&\mathsf{pecy}(\mathcal{A}_1, pt) \wedge \mathsf{pecy}(\mathcal{A}_2, pt) \wedge \\ &\forall(\ell_{\mathsf{fin}_1}, \ell_{\mathsf{fin}_2}) \in L\ \exists(\langle(\ell_1, \ell_2), \nu\rangle, t) \in \mathsf{Fin}(\mathcal{A}_1||\mathcal{A}_2) \bullet (\ell_1, \ell_2) \neq (\ell_{\mathsf{fin}_1}, \ell_{\mathsf{fin}_2})\end{aligned}$$

Since $\mathsf{pecy}(\mathcal{A}_1)$ and $\mathsf{pecy}(\mathcal{A}_2)$ they have a unique final location, let ℓ'_{fin_1} and ℓ'_{fin_2} denote their final location respectively. By definition of final configuration there exists a computation $\pi \in \mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2)$ such that $\pi_i = \langle(\ell_1, \ell_2, \nu\rangle, t)$, with $\ell_1 \neq \ell'_{\mathsf{fin}_1}$ or $\ell_2 \neq \ell'_{\mathsf{fin}_2}$, and a configuration $\pi_k = (\langle(\ell_{0_1}, \ell_{0_2}), \nu'\rangle,\ t)$ such that

$$\ldots \xrightarrow{t''} \pi_i \xrightarrow{\lambda_i} \ldots \xrightarrow{\lambda_{i+1}} \ldots \xrightarrow{\lambda_{k-1}} \pi_k$$

With $\pi_k \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)$ and $\lambda_j \in \Sigma$ for $i \leq j < k$. By Lemma 3.5 $\Gamma_1(\pi) \in \mathcal{TS}(\mathcal{A}_1)$ and $\Gamma_1(\pi)_{i'} = \langle \ell_1, \nu|_{\mathbb{X}_1}\rangle, t$, $\Gamma_1(\pi)_{k'} = \langle \ell_0, \nu'|_{\mathbb{X}_1}\rangle, t$ and $\lambda_{j'}^{\Gamma_1(\pi)} \in \Sigma_1$ for $i' \leq j' < k'$. Then $\Gamma_1(\pi)_{i'} \in \mathsf{Fin}(\mathcal{A}_1)$. Since $\mathsf{pecy}(\mathcal{A}_1)$, $\ell_1 = \ell'_{\mathsf{fin}}$, which is a contradiction. □

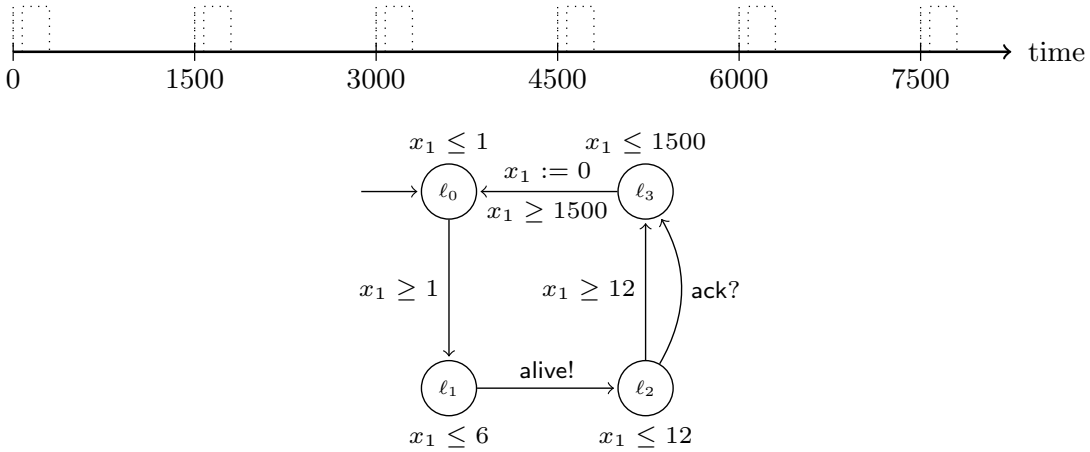


Figure 3.4: Activity of sensor 1, the activity points are at points of time where the TDMA cycle starts, and in the corresponding time slot of the sensor.

3.2 Timed Disjoint Activity

The TDMA paradigm is a technique for sharing a common media. Towards this goal information is transmitted in frames or cycles. Cycles are split into time slots. Then the components which will make use of the share media are assign to a time slot. The goal of the TDMA paradigm is to avoid collisions in the shared communication media. This fact suggest a certain independence among the components of the TDMA system. In this section, we capture and exploit the independence of actions induced by the TDMA paradigm. We formalize the independence of actions with the notions of activity and sequentialisation.

3.2.1 Activity

Transition systems induced by timed automata can perform action and delay transitions. While both types of transitions lead to changes on the system. Only action transitions may change the control location of the system, send messages, receive messages, and perform resets. Therefore, we characterize the activity of a timed automaton by the points of time at which the timed automaton performs an action transition.

Definition 3.8 (Activity)**.** *The set of* activity points $\mathsf{Active}(\mathcal{A}) \subseteq \mathbb{R}$ *of a timed automaton* $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *consists of those points in time at which action transitions take place in some computation, i.e.*

$$\mathsf{Active}(\mathcal{A}) \stackrel{\mathsf{def}}{=} \{t \in \mathbb{R} \mid \exists \pi \in \mathcal{TS}(\mathcal{A}), j \in \mathbb{N} \bullet \lambda_j \in \Sigma \wedge t(\pi_j) = t\}.$$

♢

Example 3.9. Figure 3.4 depicts the action points of time, corresponding to the sensor 1 from the fire alarm system presented in Section 1.2. Note that the activity points of time, form intervals (denoted by dotted lines), which correspond to the slot assigned to the sensor. ♢

3.2.2 Sequentialisable

The structure of TDMA based systems implies an execution ordering on its components. In general, one component communicates during his time slot. Once the time slot has elapsed. Then, the component in the next time slot communicates. We use the notions of activity and period to formalize the disjoint activity of the TDMA components in every cycle. For two periodic cyclic timed automata. The definition bellow among other conditions asserts that at every cycle, the activity of one component is strictly before the other one. If all the conditions are satisfied, we say that two timed automata are sequentialisable.

Definition 3.10 (Sequentialisable)**.** *Two timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are called* sequentialisable *if and only if*

1. *$\mathcal{A}_1$ and $\mathcal{A}_2$ have disjoint sets of clocks,*
2. *$\mathcal{A}_1$ and $\mathcal{A}_2$ are periodic cyclic with period $pt \in \mathbb{R}$, and*
3. *for each $p \in \mathbb{N}_0$,* within *the p-th period, $\mathcal{A}_1$ is active strictly before $\mathcal{A}_2$, i.e.*

$$sup(\mathsf{Active}_p(\mathcal{A}_1)) < inf(\mathsf{Active}_p(\mathcal{A}_2))$$

where $\mathsf{Active}_p(\mathcal{A}_i) \stackrel{\mathsf{def}}{=} \mathsf{Active}(\mathcal{A}_i) \cap]pt \cdot p, pt \cdot (p+1)[,\ i = 1, 2.$

♢

Note 3.11. Note that at the points of time that are multiples of the period, both automata can be active. In addition, *within* the p-th period, $p \in \mathbb{N}_0$, the activity points of two sequentialisable timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint, i.e.

$$\mathsf{Active}(\mathcal{A}_1) \cap \mathsf{Active}(\mathcal{A}_2) \cap]pt \cdot p, pt \cdot (p+1)[= \emptyset.$$

♢

Example 3.12. Consider the periodic cyclic timed automata in Figure 3.5 The time axis shows the active points of sensor 1 and sensor 2. Their activity is disjoint except at points of time which are multiples of the period. Further, note that the activity of sensor 1 precedes the activity of sensor 2 in every cycle. Therefore, sensor 1 and sensor 2 are sequentialisable. ♢

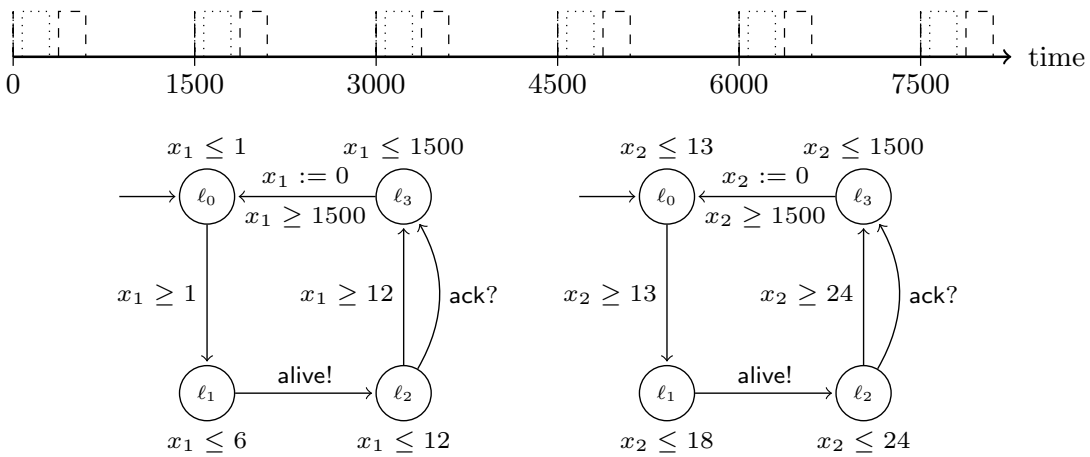


Figure 3.5: Activity of sensor 1 and sensor 2 from the fire alarm system presented in Section 1.2. The activity points are disjoint except at the points of time which are multiples of the period of the TDMA cycle. Since the automata satisfy the conditions from Definition 3.10, the automata are sequentialisable.

For timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are sequentialisable, then on each period. First, $\mathcal{A}_1$ is active and reaches its final location while $\mathcal{A}_2$ is at its initial location. Next, $\mathcal{A}_2$ is active and reaches its final location while $\mathcal{A}_1$ is at its final location. Finally, at the end of the period both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active at locations corresponding to their reset configurations. The following lemma formalizes this observations.

Lemma 3.13. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequentialisable timed automata with period pt.*

1. *For all points of time different than the integer multiples of pt and within the activity of $\mathcal{A}_1$, $\mathcal{A}_2$ is in its initial location ℓ_{0_2}, i.e.*

$$\begin{aligned}&\forall p \in \mathbb{N}_0, t \in \mathbb{R} \bullet t \in [\mathit{inf}(\mathsf{Active}_p(\mathcal{A}_1)), \mathit{sup}(\mathsf{Active}_p(\mathcal{A}_1))]\\ &\quad \Longrightarrow \forall \pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)\ \exists \langle (\ell_1, \ell_2), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2), j \in \mathbb{N}_0 \bullet\\ &\qquad \pi_j = \langle (\ell_1, \ell_2), \nu \rangle, t \wedge \ell_2 = \ell_{0_2}.\end{aligned}$$

2. *For all points of time different than the integer multiples of pt and within the activity of $\mathcal{A}_2$, $\mathcal{A}_1$ is in its final location ℓ_{fin_1}, i.e.*

$$\begin{aligned}&\forall p \in \mathbb{N}_0, t \in \mathbb{R} \bullet t \in [\mathit{inf}(\mathsf{Active}_p(\mathcal{A}_2)), \mathit{sup}(\mathsf{Active}_p(\mathcal{A}_2))]\\ &\quad \Longrightarrow \forall \pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)\ \exists \langle (\ell_1, \ell_2), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2), j \in \mathbb{N}_0 \bullet\\ &\qquad \pi_j = \langle (\ell_1, \ell_2), \nu \rangle, t \wedge \ell_1 = \ell_{\mathsf{fin}_1}.\end{aligned}$$

3. *In each computation, both, $\mathcal{A}_1$ and $\mathcal{A}_2$, are simultaneously at a restart location at integer multiples of pt, i.e.*

$$\begin{aligned}&\forall p \in \mathbb{N}_0, t \in \mathbb{R} \bullet t = pt \cdot p\\ &\implies \forall \pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)\ \exists \langle (\ell_1, \ell_2), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2), j \in \mathbb{N}_0 \bullet\\ &\qquad \pi_j = \langle (\ell_1, \ell_2), \nu \rangle, t \wedge \ell_1 \in L_{\mathsf{rst}} \wedge \ell_2 \in L_{\mathsf{rst}}.\end{aligned}$$

Proof. The lemma follows from the fact that $(\mathcal{A}_1||\mathcal{A}_2)$ is periodic cyclic with period pt and the sequentialisable assumptions, Formally:

- (1) By Theorem 3.7 it follows $\mathsf{pecy}(\mathcal{A}_1||\mathcal{A}_2)$. Therefore, at time point $pt{\cdot}p$, $(\mathcal{A}_1||\mathcal{A}_2)$ reaches a start configuration, and from there on, there are only delay transitions at time point $pt \cdot p$. Let $(c = \langle (\ell_{0_1}, \ell_{0_2}), \nu \rangle, pt \cdot p \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2))$ be such a configuration. By the sequentialisable assumption we have that $\mathit{sup}(\mathsf{Active}_p(\mathcal{A}_1)) < \mathit{inf}(\mathsf{Active}_p(\mathcal{A}_2))$ which ensures that there are not local transitions in $\mathcal{A}_2$ until time point $\mathit{sup}(\mathsf{Active}_p(\mathcal{A}_1))$. Therefore, $\ell_2 = \ell_{0_2}$ forall reachable configurations $\langle (\ell_1, \ell_2), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1||\mathcal{A}_2)$ with $t \in [\mathit{inf}(\mathsf{Active}_p(\mathcal{A}_1)), \mathit{sup}(\mathsf{Active}_p(\mathcal{A}_1))]$.

- (2) By Theorem 3.7 it follows $\mathsf{pecy}(\mathcal{A}_1||\mathcal{A}_2)$. Therefore, at time point $pt \cdot (p+1)$, $\mathcal{A}_1||\mathcal{A}_2$ reaches a start configuration. Let π_k be such a configuration. Then there exist a final configuration $\pi_i = (\langle (\ell_1', \ell_2'), \nu' \rangle, pt \cdot (p+1))$. which reaches π_k via action transitions. and by definition $\lambda_{i-1} \in \mathbb{R}$. Then there exist i' such that $\Gamma_1(\pi)_{i'} = \langle \ell_1', \nu'|_{\mathbb{X}_1} \rangle, pt{\cdot}(p{+}1)$ and $\ell_1' = \ell_{\mathsf{fin}_1}$. By the sequentialisable assumption we have $\mathit{sup}(\mathsf{Active}_p(\mathcal{A}_1)) < \mathit{inf}(\mathsf{Active}_p(\mathcal{A}_2))$. Therefore, there is no local transitions in $\mathcal{A}_1$ for any $t \in [\mathit{inf}(\mathsf{Active}_p(\mathcal{A}_2)), \mathit{sup}(\mathsf{Active}_p(\mathcal{A}_2))]$. Thus, $\ell_1 = \ell_{\mathsf{fin}}$.

- (3) Let $\beta_i(\langle (\ell_1, \ell_2), \nu \rangle, t) = \langle (\ell_i), \nu|_{\mathbb{X}_i} \rangle, t$ be a i-th configuration projection. By Theorem 3.7 it follows that $\mathsf{pecy}(\mathcal{A}_1||\mathcal{A}_2)$. Consider case when $p = 0$. There are only delay transitions at time point 0. Thus, we are at c_0 and $\ell_{0_1} \in L_{\mathsf{rst}_1}$ and $\ell_{0_2} \in L_{\mathsf{rst}_2}$. Consider case when $p \neq 0$. There exists $i, j, k \in \mathbb{N}$ such that $\pi_i \in \mathsf{Fin}(\mathcal{A}_1||\mathcal{A}_2)$, $\pi_k \in \mathsf{Start}(\mathcal{A}_1||\mathcal{A}_2)$, and if $i \leq j < k$ then $\lambda_j \in \Sigma$. Then there exist $i', j', k' \in \mathbb{N}$ such that $\beta_1(\pi_i) = \Gamma_1(\pi)_{i'}$, $\beta_1(\pi_k) = \Gamma_1(\pi)_{k'}$ with $\Gamma_1(\pi)_{i'} \in \mathsf{Fin}(\mathcal{A}_1)$, $\Gamma_1(\pi)_{k'} \in \mathsf{Start}(\mathcal{A}_1)$ and if $i' \leq j' < k'$ then $\lambda_{k'}^{\Gamma_1(\pi)} \in \Sigma_1$. Then by definition of L_{rst_1} it follows that $\ell_1 \in L_{\mathsf{rst}_1}$.

□

3.3 Concatenation of Periodic Cyclic Timed Automata

For sequentialisable automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with period pt Lemma 3.13 suggests that for time points different than $pt \cdot p$ for some p, it is not necessary to compute the product of the locations on $\mathcal{A}_1$ and $\mathcal{A}_2$. The following concatenation operator exploits this fact and computes the product of locations only if both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active. i.e. at time points $pt \cdot p$.

Definition 3.14 (Concatenation)**.**
Let $\mathcal{A}_1 = (L_1, \Sigma_1, \mathbb{X}_1, I_1, E_1, \ell_{0_1})$ *and* $\mathcal{A}_2 = (L_2, \Sigma_2, \mathbb{X}_2, I_2, E_2, \ell_{0_2})$ *be sequentialisable timed automata with period* $pt \in \mathbb{R}$*. Let* ℓ_{fin_i} *denote the final location and let* L_{rst_i} *denote the set of restart locations of automaton* $\mathcal{A}_i$*,* $i \in \{1, 2\}$*.*

The concatenation *of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *yields the timed automaton*

$$\mathcal{A}_1 \cdot \mathcal{A}_2 \stackrel{\mathsf{def}}{=} (L, \Sigma_1 \cup \Sigma_2, \mathbb{X}_1 \cup \mathbb{X}_2, I, E, \ell_0)$$

where

- $L = (L_1 \times \{\ell_{0_2}\}) \cup (\{\ell_{\mathsf{fin}_1}\} \times L_2) \cup (L_{\mathsf{rst}_1} \times L_{\mathsf{rst}_2})$
- $I(\ell_1, \ell_2) = I_1(\ell_1) \wedge I_2(\ell_2)$*,* $\ell_1 \in L_1, \ell_2 \in L_2$*,*
- $E = \{((\ell_1, \ell_2), \alpha, \varphi_1, Y_1, (\ell_1', \ell_2)) \mid (\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1 \wedge \ell_2 \in L_{\mathsf{rst}_2}\}$
  $\cup \{((\ell_1, \ell_2), \alpha, \varphi_2, Y_2, (\ell_1, \ell_2')) \mid (\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2 \wedge \ell_1 \in L_{\mathsf{rst}_1}\}$*, and*
- $\ell_0 = (\ell_{0_1}, \ell_{0_2})$*.*

♢

By having a closer look into the concatenation operator, we observe that the definition of the invariant function and the initial location is similar to the one of the parallel product. However, the resulting sets of locations and edges are different. In the parallel composition operator the set of locations corresponds to the Cartesian product on the sets of locations of every component. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are sequentialisable Lemma 3.13 suggest that there are pairs of locations which are not reachable. By reachable we mean that these locations will not be part of a configuration in a computation. Therefore, the set of locations resulting from the application of the concatenation operator discards by construction all unreachable locations. The set of edges is the union of two sets. The first set corresponds to all the edges from $\mathcal{A}_1$ while the other automaton is idle. The second set corresponds to the edges in $\mathcal{A}_2$ while the other automaton is idle. As a result, the automaton resulting from the application of the concatenation operator has a reduced number of edges and locations.

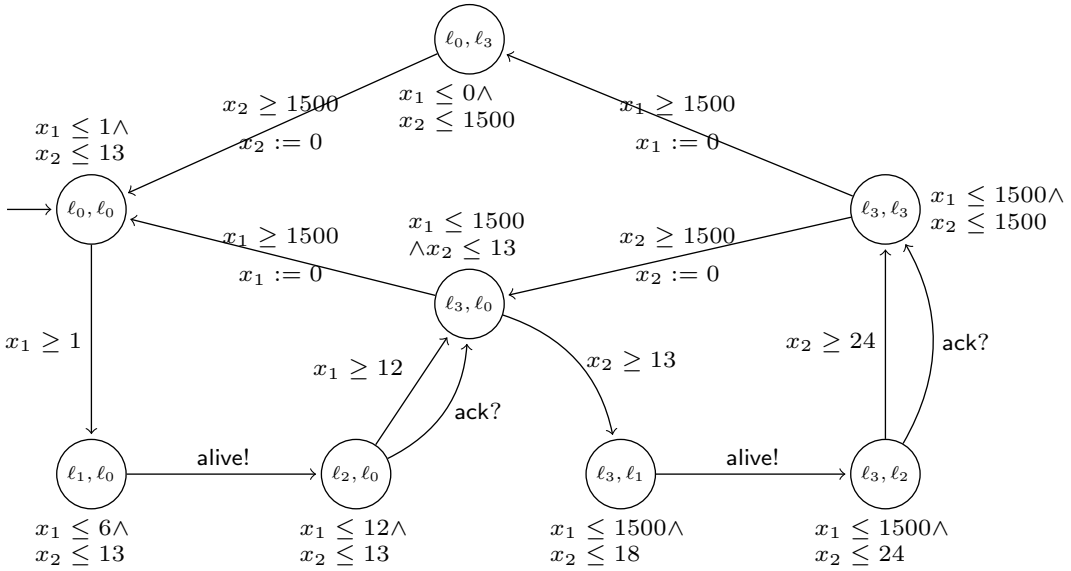


Figure 3.6: Concatenation operator applied on sensor 1 and sensor 2 from the fire alarm system presented in Section 1.2. Note that the sensors have disjoint activity and are sequentialisable. In addition, note that the resulting automaton corresponds to the reachable part obtained by computing the parallel product of the two sensors.

Example 3.15. Consider sensor 1 and sensor 2 from Figure 3.5 from the fire alarm system presented in Section 1.2. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ denote sensor 1 and sensor 2 respectively. Then, by the Definition 3.1 of start, final, and restart configurations we have that $L_{\mathsf{rst}_1} = \{\ell_0, \ell_3\}$, $L_{\mathsf{rst}_2} = \{\ell_0, \ell_3\}$, $\ell_{\mathsf{fin}_1} = \ell_3$, and $\ell_{\mathsf{fin}_2} = \ell_3$. In addition, note that $\mathcal{A}_1$ and $\mathcal{A}_2$ satisfy the sequentialisable conditions stated in Definition 3.10. Now, applying the concatenation operator on $\mathcal{A}_1$ and $\mathcal{A}_2$ we obtain the automaton in Figure 3.6. $\diamondsuit$

Consider the automaton in Figure 3.3 corresponding to the reachable automaton obtained by application of the parallel product operator on sensor 1 and sensor 2 from the fire alarm system presented in Section 1.2. Now consider the automaton in Figure 3.6 corresponding to the concatenation of sensor 1 and sensor 2. We observe that both automata are identical. In the following, we formalize the relation between the automata obtained by application parallel composition and concatenation.

Definition 3.16 (Bisimulation)**.** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be timed automata and*

$$\mathcal{TS}_i(\mathcal{A}_i) = (\mathit{Conf}(\mathcal{A}_i), \mathbb{R} \cup \Sigma_i, \{\xrightarrow{\lambda^i} | \ \lambda^i \in \mathbb{R} \cup \Sigma_i\}, C_{0_i})$$

the corresponding labeled transition systems. A relation $\mathcal{R} \subseteq \mathit{Conf}(\mathcal{A}_1) \times \mathit{Conf}(\mathcal{A}_2)$ is called bisimulation *of $\mathcal{A}_1$ and $\mathcal{A}_2$ if and only if it satisfies the following conditions.*

1. $\forall c_1 \in C_{0_1}\ \exists c_2 \in C_{0_2} \bullet (c_1, c_2) \in \mathcal{R}$ *and* $\forall c_2 \in C_{0_2}\ \exists c_1 \in C_{0_1}) \bullet (c_1, c_2) \in \mathcal{R}$

2. *for all* $(c_1 = (\langle \ell_1, \nu_1 \rangle, t_1),\ c_2 = (\langle \ell_2, \nu_2 \rangle, t_2)) \in \mathcal{R}$,

   a) $\nu_1 = \nu_2,\ t_1 = t_2$,

   b) $\forall c_1 \xrightarrow{\lambda^1} c_1'\ \exists c_2 \xrightarrow{\lambda^2} c_2 \bullet (c_1', c_2') \in \mathcal{R}$

   c) $\forall c_2 \xrightarrow{\lambda^2} c_2'\ \exists c_1 \xrightarrow{\lambda^1} c_1' \bullet (c_1', c_2') \in \mathcal{R}$.

$\mathcal{A}_1$ *is called bisimilar to* $\mathcal{A}_2$ *iff there exists a bisimulation of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$. ♢

For sequentialisable timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$, the implications of Lemma 3.13, and the definition of the concatenation operator, imply that the transition system $\mathcal{TS}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ corresponds to the reachable part of $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$. The following theorem states that for sequentialisable timed automata. The transition system induced by the automaton corresponding to the application of the concatenation operator is bisimilar to the transition system induced by the automaton corresponding to the application of the parallel composition operator.

Theorem 3.17. *Let* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *be sequentialisable timed automata. Then* $\mathcal{TS}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ *is bisimilar to* $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$.

Proof. Construct $\mathcal{R}$ as

$$\mathcal{R} = \{(c, c) \in \mathit{Conf}(\mathcal{A}_1 \cdot \mathcal{A}_2) \times \mathit{Conf}(\mathcal{A}_1 || \mathcal{A}_2) \mid c \in \mathit{Conf}(\mathcal{A}_1 \cdot \mathcal{A}_2)\}$$

Condition (1). Let $c = \langle \ell_{0_1}, \ell_{0_2}, 0 \rangle, 0$. We have $c_{0_1} = c_{0_2} = \{c\}$. Therefore, $(c, c) \in \mathcal{R}$.
Condition (2). Let $(c_1, c_2) \in \mathcal{R}$, by construction of $\mathcal{R}$ we know that $c_1 = c_2$. Therefore, condition (2.a) is satisfied. For condition (2.b) there is a transition in $\mathcal{A}_1 \cdot \mathcal{A}_2$. There is either a delay transition or an action transition in $\mathcal{TS}(\mathcal{A}_1 \cdot \mathcal{A}_2)$. We distinguish two cases:

- Delay transitions: then there is a transition

  $$c_1 = \langle (\ell_1, \ell_2), \nu_1 \rangle, t_1 \xrightarrow{t'} \langle (\ell_1, \ell_2), \nu_1 + t' \rangle, t_1 + t' = c_1'$$

  such that $\nu_1 + t'' \models I_1(\ell_1) \wedge I_2(\ell_2)$ for all $t'' \in [0, t']$. Now let $c_2' = \langle (\ell_1, \ell_2), \nu_1 + t' \rangle, t_1 + t'$. Then $c_2 \xrightarrow{t'} c_2'$ since $\nu_1 + t'' \models I_1(\ell_1) \wedge I_2(\ell_2)$ for all $t'' \in [0, t']$ and $(c_1', c_2') \in \mathcal{R}$ since $c_2' = c_1'$.

- Action transitions: first consider there is a transition in $\mathcal{A}_1$

  $$c_1 = \langle (\ell_1, \ell_2), \nu_1 \rangle, t_1 \xrightarrow{\alpha} \langle (\ell_1', \ell_2), \nu_1' \rangle, t_1 = c_1'$$

then there is an edge $e = (\ell_1, \alpha, \varphi, Y, \ell_1') \in E_1$ such that $\nu_1 \models \varphi$, $\nu_1' \models v_1[Y := 0]$ and $\nu_1' \models I_1(\ell_1)$. Now let $c_2 \xrightarrow{\alpha} c_2'$ by the action transition induced by e with $c_2' = \langle(\ell_1', \ell_2), \nu_1'\rangle, t_1$. Clearly $(c_1', c_2') \in \mathcal{R}$ since $c_1' = c_2'$. The case for the second component $\mathcal{A}_2$ can be shown analogously.

Condition (2.c). There is a transition in $\mathcal{A}_1||\mathcal{A}_2$. There is either a delay transition or an action transition in $\mathcal{A}_1||\mathcal{A}_2$. Delay transitions are treated as above for condition (2.b). Thus, we consider now only action transitions. We distinguish three cases. when $\mathcal{A}_1$ is active, i.e. the points in $\mathsf{Active}(\mathcal{A}_1)$, when $\mathcal{A}_2$ is active and when both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active. i.e. at time point *pt*.

- Time points in $\mathsf{Active}(\mathcal{A}_1)$. Let $t \in \mathsf{Active}(\mathcal{A}_1)$. There are action transitions in $\mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2)$ corresponding to some edge in E_1. Thus we have

  $$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t \xrightarrow{\alpha} \langle(\ell_1', \ell_2), \nu_2'\rangle, t = c_2'$$

  then by definition of $||$ and the fact that $t \in \mathsf{Active}(\mathcal{A}_1)$ we know that there exist an edge $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ such that $\nu_2 \models \varphi$, $\nu_2' = \nu_2[Y := 0]$ and $\nu_2' \models I_1(\ell_1') \wedge I_2(\ell_2)$. By Lemma 3.13 we know that $\ell_2 = \ell_{0_2}$ and by definition of $\cdot$ we know that there exist an edge $e = ((\ell_1, \ell_2), \alpha, \varphi_1, Y_1, (\ell_1', \ell_2))$ in $\mathcal{A}_1 \cdot \mathcal{A}_2$ such that $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ and $\ell_2 \in L_{\mathsf{rst}_2}$. Now complete $c_1 = c_2 \xrightarrow{\alpha} c_1'$ with $c_1' = \langle(\ell_1', \ell_2), \nu_1'\rangle, t_1'$ with $\nu_1' = \nu_2' = \nu_2[Y := 0]$. It follows that $\nu_1' \models I_1(\ell_1') \wedge I_2(\ell_2)$. Therefore, $c_1' \in \mathit{Conf}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ and $(c_1', c_2') \in \mathcal{R}$.

- Time points in $\mathsf{Active}(\mathcal{A}_2)$. There are action transitions in $\mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2)$ corresponding to some edge in E_2. Thus we have

  $$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t \xrightarrow{\alpha} \langle(\ell_1, \ell_2'), \nu_2'\rangle, t = c_2'$$

  then by definition of the parallel composition operator $||$ and the fact that $t \in \mathsf{Active}(\mathcal{A}_2)$ we know that there exist an edge $(\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2$ such that $\nu_2 \models \varphi$, $\nu_2' = \nu_2[Y_2 := 0]$ and $\nu_2' \models I_1(\ell_1) \wedge I_2(\ell_2')$. By Lemma 3.13 we know that $\ell_1 = \ell_{\mathsf{fin}_1}$, $\ell_1 \in L_{\mathsf{fin}_1}$, and by definition of the concatenation operator $\cdot$. We know that there exist an edge $e = ((\ell_1, \ell_2), \alpha, \varphi_2, Y_2, (\ell_1, \ell_2'))$ in $\mathcal{A}_1 \cdot \mathcal{A}_2$ such that $(\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2$ and $\ell_1 \in L_{\mathsf{fin}_1}$. Now complete $c_1 = c_2 \xrightarrow{\alpha} c_1'$ with $c_1' = \langle(\ell_1, \ell_2'), \nu_1'\rangle, t_1'$ with $\nu_1' = \nu_2' = \nu_2[Y_2 := 0]$. It follows that $\nu_1' \models I_1(\ell_1) \wedge I_2(\ell_2')$. Therefore, $c_1' \in \mathit{Conf}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ and $(c_1', c_2') \in \mathcal{R}$.

- Time point $pt \cdot p$ for some $p \in \mathbb{N}$. Then, both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active. Therefore, local transition and synchronization transitions can occur in $\mathcal{A}_1$ and $\mathcal{A}_2$. Without loss of generality, consider there is a local transition in $\mathcal{A}_1$. There are action transitions in $\mathcal{TS}(\mathcal{A}_1||\mathcal{A}_2)$ corresponding to some edge in E_1. Thus we have

$$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t \xrightarrow{\alpha} \langle(\ell_1', \ell_2), \nu_2'\rangle, t = c_2'$$

  then by definition of the parallel composition operator $\|$ and the fact that $t \in \mathsf{Active}(\mathcal{A}_1)$. We know that there exist an edge $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ such that $\nu_2 \models \varphi$, $\nu_2' = \nu_2[Y := 0]$ and $\nu_2' \models I_1(\ell_1') \wedge I_2(\ell_2)$. By Lemma 3.13 we know that $\ell_2 \in L_{\mathsf{rst}_2}$, and by definition of the concatenation operator $\cdot$ we know that there exist an edge $e = ((\ell_1, \ell_2), \alpha, \varphi_1, Y_1, (\ell_1', \ell_2))$ in $\mathcal{A}_1 \cdot \mathcal{A}_2$ such that $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ and $\ell_2 \in L_{\mathsf{rst}_2}$. Now complete $c_1 = c_2 \xrightarrow{\alpha} c_1'$ with $c_1' = \langle(\ell_1, \ell_2'), \nu_1'\rangle, t_1'$ with $\nu_1' = \nu_2' = \nu_2[Y_2 := 0]$. It follows that $\nu_1' \models I_1(\ell_1) \wedge I_2(\ell_2')$. Therefore, $c_1' \in \mathit{Conf}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ and $(c_1', c_2') \in \mathcal{R}$. This concludes our proof.

□

The class of periodic cyclic timed automata is closed under the application of the parallel composition operator. In the following theorem we show that the class of periodic cyclic timed automata is closed under the application of the concatenation operator. This property is important since it allows to concatenate automata in a compositional manner. The theorem follows from the fact that for two sequentialisable timed automata, the transition systems obtained by application of the parallel composition and concatenation operators are bisimilar.

Theorem 3.18. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequentialisable timed automata with period pt. Then $\mathcal{A}_1 \cdot \mathcal{A}_2$ is periodic cyclic with period pt.*

Proof. For sequentialisable timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$ with period pt. By Theorem 3.7 we know that $\mathcal{TS}(\mathcal{A}_1\|\mathcal{A}_2)$ is periodic cyclic with period pt. By Theorem 3.17 $\mathcal{TS}(\mathcal{A}_1\|\mathcal{A}_2)$ and $\mathcal{TS}(\mathcal{A}_1 \cdot \mathcal{A}_2)$ are bisimilar. Therefore, the start configurations of $\mathcal{TS}(\mathcal{A}_1\|\mathcal{A}_2)$ are in $\mathcal{TS}(\mathcal{A}_1 \cdot \mathcal{A}_2)$. □

For sequentialisable timed automata, the transition systems obtained by application of parallel composition and concatenation are bisimilar. Therefore, the behavior of the system is maintained and properties are preserved. This is expressed in the the following corollary.

Corollary 3.19 (Reachability Properties)**.** Reachability properties *are preserved under bisimulation, i.e. given bisimilar timed automata* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *and a state assertion* φ*, i.e. an expression over clock constraints and locations, we have*

$$(\exists \pi \in \mathcal{TS}(\mathcal{A}_1)\ \forall j \in \mathbb{N}_0 \bullet \pi_j \models \varphi) \iff (\exists \pi \in \mathcal{TS}(\mathcal{A}_2)\ \forall j \in \mathbb{N}_0 \bullet \pi_j \models \varphi)$$
$$(\forall \pi \in \mathcal{TS}(\mathcal{A}_1)\ \forall j \in \mathbb{N}_0 \bullet \pi_j \models \varphi) \iff (\forall \pi \in \mathcal{TS}(\mathcal{A}_2)\ \forall j \in \mathbb{N}_0 \bullet \pi_j \models \varphi).$$

3.4 Complexity

In this section we show that for sequentialisable timed automata, the use of the concatenation operator leads to quadratic speed ups compared to the use of the parallel composition operator. The main idea behind this gain, is that for sequentialisable timed automata, at a given point of time, only one automaton is active. Therefore, applying the concatenation operator will produce an automaton with a reduced number of edges and locations. In particular, unreachable edges are not present. By unreachable edges we refer to edges which do not occur in any computation. These unreachable edges have a cost in verification, since guards have to be evaluated to see if an edge can be taken.

In the following, for a configuration we define the notion of outgoing and enabled edges. Outgoing edges are the edges starting at a given configuration. These edges do not necessarily form part of a computation. On the contrary, enabled edges are edges with satisfiable guards and the effects on the valuations satisfy the destination invariants. Therefore, enabled edges induce transitions in some computation paths.

Definition 3.20 (Enabled Edges)**.** *Let* $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *be a timed automaton and* $c \in \mathit{Conf}(\mathcal{A})$ *a configuration.*

We use $\mathit{out}(c)$ *to denote the set of* outgoing *edges in* c*, i.e. the edges* $e = (\ell, \alpha, \varphi, Y, \ell') \in E$ *where* $\ell = \ell(c)$*.*

Edge e *is called* enabled *if and only if its guard is satisfied and the effect of resets satisfies the clock constraint of the destination of* e*, i.e. if* $\nu(c) \models \varphi$ *and* $\nu(c)[Y := 0] \models I(\ell')$*. We use* $\mathit{enab}(c)$ *to denote the set of edges enabled in* c*.*

In the following we describe some facts and notation about enabled edges in an automaton resulting from the parallel composition of a number of timed automata.

Note 3.21. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be timed automata with pairwise disjoint edge sets and let $c = \langle (\ell_1, \ldots, \ell_n), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1 \| \ldots \| \mathcal{A}_n)$ be a configuration.

1. Enabled edges are in particular outgoing, i.e. $\mathit{enab}(c) \subseteq \mathit{out}(c)$.

2. The set of outgoing edges in the parallel composition is determined by the components, i.e.

$$out(c) = \bigcup_{1 \leq i \leq n} out(\langle \ell_i, \nu|_{\mathbb{X}_i} \rangle, t), enab(c) = \bigcup_{1 \leq i \leq n} enab(\langle \ell_i, \nu|_{\mathbb{X}_i} \rangle, t),$$

thus (with disjoint edge sets)

$$|out(c)| = \Sigma_{1 \leq i \leq n} |out(\langle \ell_i, \nu|_{\mathbb{X}_i} \rangle, t)|, |enab(c)| = \Sigma_{1 \leq i \leq n} |enab(\langle \ell_i, \nu|_{\mathbb{X}_i} \rangle, t)|.$$

For a timed automaton obtained by parallel composition of sequentialisable timed automata. The number of outgoing edges is much larger than the number of enabled edges. This is because the parallel composition operator do not exploit the sequentialisable assumptions. Most of this unnecessary edges will be removed by using the concatenation operator. Since, outgoing edges are evaluated, a reduction on the number of outgoing edges yields a reduction on time complexity. This reduction goes from quadratic to linear time as the following lemma shows.

Lemma 3.22. *Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be sequentialisable timed automata with period pt with disjoint edge sets and with exactly one outgoing edge per location.*

1. *Let $c \in Conf(\mathcal{A}_1 \| \ldots \| \mathcal{A}_n)$ be a configuration of the parallel composition of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where the time-stamp is not an integer multiple of the period pt, i.e. where $\nexists p \in \mathbb{N}_0 \bullet t(n) = p \cdot pt$.*

   Then $|out(c)| = n$ and $|enab(c)| = 1$.

2. *Let $c \in Conf(\mathcal{A}_1 \cdot \ldots \cdot \mathcal{A}_n)$ be a configuration of the concatenation of $\mathcal{A}_1, \ldots, \mathcal{A}_n$ where the time-stamp is not an integer multiple the period pt.*

   Then $|out(c)| = |enab(c)| = 1$.

Given a number of sequentialisable timed automata with period pt. Lemma 3.22 states a comparison between the system obtained by parallel composition and by concatenation operator at points of time which are not multiples of the period.

We observe that for the system obtained by parallel composition, the number of outgoing edges is n where n is the number of components. Further, the number of enabled edges is 1. This means that at this point of time the model checker needs to perform additional $n - 1$ checks on the guards and invariants of these edges.

In addition, we observe that for the system obtained using the concatenation operator. The number of outgoing edges is the same as the number of enabled edges. Therefore, at this point only required computations are executed.

The following example, shows that in the case of the fire alarm system presented in Section 1.2, quadratic speed ups are achieved by using the concatenation operator.

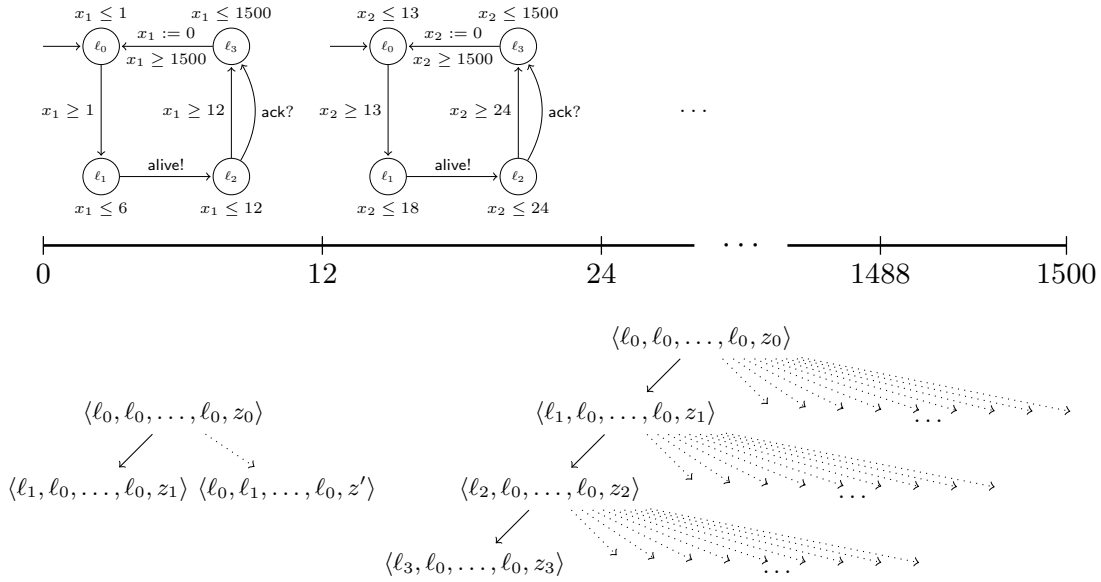


Figure 3.7: A reduced number of edges. Top, a number of sensors which are sequentialisable. Bottom left, checking the enabledness of an edge which belongs to sensor two, where sensor one is at his corresponding time slot. Bottom right, after the TDMA cycle there where a number of unneeded enableness checks. The unnecessary enableness check are denoted by dotted lines.

Example 3.23. (A reduced number of edges) Consider Figure 3.7, the figure shows a time axis and a number N of sensors $\mathcal{A}_1, \ldots \mathcal{A}_N$ from the fire alarm system presented in Section 1.2. Every sensor is assigned to its corresponding time slot. The sensors are periodic cyclic and sequentialisable.

Now consider the transition system $\mathcal{TS}(\mathcal{A}_1 \| \ldots \| \mathcal{A}_N)$. Figure 3.7 bottom-left, shows a possible computation from the the model checker at configuration $\langle \ell_0, \ell_0, \ldots, \ell_0, z_0 \rangle, t$ for some $t \in]0, 12[$. Since there is an edge from sensor 2 from location ℓ_0 to location ℓ_1, the model checker evaluates this edge. This is unnecessary, because sensor 1 is at its time slot. The doted line indicates the unnecessary evaluation of this edge. At the bottom-right of Figure 3.7 there is a possible computation tree from the the model checker. Note the number of unnecessary edge evaluations, denoted by dotted lines.

For every sensor, the model checker evaluates the other $(N-1)$ sensors. Thus at every point of time, the model checker performs $N \times (N-1)$ comparisons. These unreachable edges are not present in $\mathcal{A}_1 \cdot \ldots \cdot \mathcal{A}_N$. Therefore, at every point of time, the model checker

performs N comparisons, i.e. we achieve a quadratic speed up. ♢

3.5 Related Work

Since the cost of model checking for a network of timed automata increases exponentially in the number of components [129], much research has been directed towards techniques that demonstrate a potentially exponential speedup in interesting applications (see, e.g. [16, 54, 72]).

Partial order reduction methods for timed systems have been subject to several publications [25, 52, 98, 72]. This approach is complementary to ours. By using our method, interleavings are possible at the points of time which are multiples of the TDMA period. If a great number of components are involved, interleavings may cause a state space explosion which might hinder the verification task. Partial order reduction might come useful in handling the interleavings for sequentialisable timed automata, in particular at points of time which are multiples of the period.

On [119] the result: the union of all zones reached by different interleavings of the same set of transitions is convex, is presented. This result is then applied in the reachability analysis by merging zones. As a result the state explosion induced by the interleaving semantics can be avoided. Note that this result is complementary to ours. Because, at points of time which are multiples of the period. We allow full interleavings. At these points of time this technique could came useful.

Another important technique for reducing the state space, is the technique of active clock reduction of [54] and its generalization in [16]. These techniques may seem relevant in this context. These techniques are however complementary. That is, after a successful clock optimization following [54]. Our technique will still improve the verification time for sequentialisable timed automata, because unnecessary edges have not been removed. In fact, when we present our experimental evaluation our experiments compare to an optimized model which has only one clock and unnecessary edges, we obtain quadratic speed ups. The non optimized model will have over 100 clocks.

In [63, 86, 106] Communication-Closed Layers and timed automata are studied. The approach presented in [106] and ours are complementary. The main differences are that the approach in [106] is action based, whereas our approach is time based. In addition, we consider cyclic timed automata, whereas in [106], automata can not perform actions after reaching their corresponding final location.

Chapter 4

Syntactic Optimizations for TDMA Systems

Contents

In the previous chapter, we have introduced the class of periodic cyclic timed automata. This class has the property that start configuration is visited infinitely often within regular periods of time. We then introduce the notion of sequentialisability. If two automata are sequentialisable, then the concatenation operator could be applied. Applying the concatenation operator leads to quadratic speed ups.

The results of the previous chapter are interesting from a theoretical point of view. However, in practice they are difficult to implement. As an example, consider an timed automaton is given. How could we detect if the automaton is periodic cyclic? or if we are given two timed automata. How can we detect if they sequentialisable?. Answering these question may be as hard as performing the model checking task. In this chapter, we address these questions and propose a solution. Our solution consist in providing a number of syntactic patterns and operators which exploit particular properties of TDMA based systems.

We present the syntactic class of timed automata which we call sequential timed automata. By construction, this class ensures the corresponding automaton to be periodic cyclic. Sequential timed automata syntactically describe relevant properties of periodic cyclic timed automata. They describe; the start and final points of their activity, their initial and final locations, and their period. As a result, the sequentialisable conditions, can be checked syntactically in the given automata.

Next, we present the concept of overclocks. For TDMA based systems with several components, it is natural to consider that every component has his own independent clock. However, we note that from an observer point of view. The components are

behaving synchronously with respect to one master clock. This observation lead us to the definition of overclocks. Since, the clocks of the TDMA components have a similar behavior, we show how to simplify these clocks by replacing them by an overclock. Our work on overclocks raised a successful simplification of equivalence classes on clocks. This special clocks are called quasi-equal clocks, and have been subject of several publications [76, 101, 77].

Finally, we introduce a sequential composition operator on sequential timed automata. For sequentialisable timed automata, the sequential composition operator further improves the concatenation operator presented in the previous chapter. For sequentialisable timed automata, at points of time which are multiples of its period, all automata are allowed to be active. At these points of time, the interleaving semantics of timed automata induce an exponential number of states. The use of sequential timed automata and the sequential composition operator will hinder the state explosion, by performing a clock simplyfication. In addition, the sequential composition operator will remove unnecessary edges, which further improves the verification times.

For TDMA based systems the use of, sequential timed automata, overclocks, and the corresponding sequential composition operator. Improve the verification times from exponential to linear on the number of components. This is the case for the fire alarm system that we presented in Chapter 1, Section 1.2 and published in [58, 100].

Contributions

The key technical contributions that are described in this chapter are summarized as follows:

- We introduce a class of timed automata. We call this class *sequential timed automata*. We show that this class is a subset of the class of of periodic cyclic timed automata. In particular, this class of automata, syntactically describes relevant behavioral properties. e.g. start and final points of activity.
- We introduce the notion of *overclocks*. We indentify a class of clocks which have the property that they are almost equal except at some points of time and certain locations. These clocks can be replaced by an overclock, which drastically reduces the state space while preserving most of the behavior.
- We introduce a syntactic *sequential composition* operator on sequentialisable sequential timed automata. This operator, further increases the efficiency of the concatenation operator presented in the previous chapter. This operator performs a clock simplification by means of overclocks and removes unnecessary edges. The use of the sequential composition operator avoids the state explosion induced by the interleaving semantics of timed automata.
- For sequentialisable sequential timed automata. We show that, the relation between a system composed using the parallel composition operator and a system composed using the sequential composition operator, is a weak-bisimulation relation. Further, we show that the verification costs drop from exponential to linear.

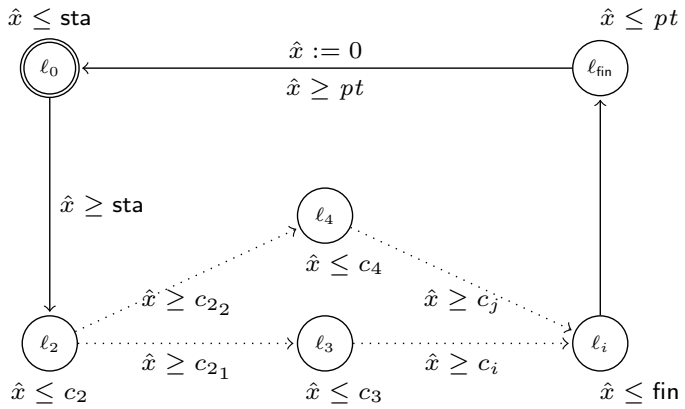


Figure 4.1: A syntactical pattern for sequential timed automata.

4.1 Sequential Timed Automata

As the decision, whether a given pair of timed automata are sequentialisable, is in general at least as difficult as the model checking task and our goal is to increase the applicability of model checking for verifying TDMA based systems. The properties of periodic cyclic timed automata and the sequentialisable assumptions, have to be effectively and automatically recognizable by a verification tool.

Toward this goal, we provide a syntactical class of timed automata. We call this class sequential timed automata. The class is given by a syntactical pattern such that instances of the pattern are periodic cyclic timed automata. See Definition 3.3. The signature of sequential timed automaton syntactically specify important properties of periodic cyclic timed automata. It specifies, the start and final points of activity, the start and final configurations, and the period of the automata. Further, sequential timed automata impose two additional constraints. There is a final edge, this edge is taken at the end of the TDMA cycle. There is a master clock, this clock is in keeping track of the TDMA cycle. The following definition formalizes the notion of sequential timed automata.

Definition 4.1 (Sequential Timed Automaton)**.** *A* sequential timed automaton *(STA) is a tuple*

$$\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0, \ell_{\mathsf{fin}}, \mathsf{sta}, \mathsf{fin}, pt, e_{\mathsf{fin}}, \hat{x})$$

where $\mathcal{A}_0 \stackrel{\mathsf{def}}{=} (L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *is a timed automaton,* ℓ_{fin} *is a final location,* $\mathsf{sta}, \mathsf{fin} \in \mathbb{Q}_0^+$ *are* start *and* final time*,* $pt \in \mathbb{Q}_0^+$ *is a period,* $e_{\mathsf{fin}} \in E$ *is an edge of the form* $(\ell_{\mathsf{fin}}, \varnothing, \hat{x} \geq pt, Y \cup \{\hat{x}\}, \ell_0)$ $\hat{x} \in \mathbb{X}$ *is a master clock, which satisfies the following syntactical constraints:*

- *the start time is positive and strictly smaller than the final time, which is strictly*

smaller than the period, i.e.

$$0 < \mathsf{sta} \wedge \mathsf{sta} < \mathsf{fin} \wedge \mathsf{fin} < pt, \tag{saActive}$$

- *the initial location is left if $\hat{x}$ reaches the start time, i.e.*

$$I(\ell_0) = \hat{x} \leq \mathsf{sta}, \tag{saStart}$$

$$\forall(\ell, \alpha, \varphi, Y, \ell') \in E \bullet \ell = \ell_0 \implies \varphi = \hat{x} \geq \mathsf{sta}, \tag{saStartTime}$$

- *locations connected by an edge to the final location are only assumed until $\hat{x}$ reaches the final time, i.e.*

$$\forall(\ell, \alpha, \varphi, Y, \ell') \in E \bullet \ell' = \ell_{\mathsf{fin}} \Rightarrow I(\ell) = \hat{x} \leq \mathsf{fin}, \tag{saFinalTime}$$

- *the final location is only assumed until $\hat{x}$ reaches the period, i.e.*

$$I(\ell_{\mathsf{fin}}) = \hat{x} \leq pt, \tag{saPeriod}$$

- *the master clock is reset exactly on edge e_{fin}, i.e.*

$$\forall(\ell, \alpha, \varphi, Y, \ell') \in E \bullet \hat{x} \in Y \Rightarrow (\ell, \alpha, \varphi, Y, \ell') = e_{\mathsf{fin}}, \tag{saOneReset}$$

- *ℓ_{fin} and ℓ_0 are connected exactly by edge e_{fin}, i.e.*

$$\forall(\ell, \alpha, \varphi, Y, \ell') \in E \bullet \ell = \ell_{\mathsf{fin}} \wedge \ell' = \ell_0 \implies (\ell, \alpha, \varphi, Y, \ell') = e_{\mathsf{fin}}, \tag{saOneFin}$$

and the following semantical constraint:

- *whenever the initial location is assumed, the final location is finally reached, i.e.*

$$\begin{aligned} &\forall \pi \in \mathcal{TS}(\mathcal{A}_0), j \in \mathbb{N}_0, \langle \ell, \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_0) \bullet \pi_j = \langle \ell, \nu \rangle, t \wedge \ell = \ell_0 \\ &\quad \implies \exists k \in \mathbb{N}_0, \langle \ell', \nu' \rangle, t' \in \mathit{Conf}(\mathcal{A}_0) \bullet \\ &\qquad k \geq j \wedge \pi_k = \langle \ell', \nu' \rangle, t' \wedge \ell' = \ell_{\mathsf{fin}} \end{aligned} \tag{saCyclic}$$

♢

Given a sequential timed automaton $\mathcal{A}$. Note that the syntactical constraints can be proven by syntactically inspecting the given sequential timed automaton. These checks can be performed in $\mathcal{O}(|\mathcal{A}|)$. For the semantical check of saCyclic. Let $\mathcal{A}'$ be the automaton obtained by replacing synchronization transitions in $\mathcal{A}$ by internal transitions. Then, saCyclic can be checked for $\mathcal{A}'$ in a model checker.

Figure 4.1 depicts a purely syntactically restricted template which ensures the instances to be sequential timed automata. The following theorem, states that sequential timed automata are periodic cyclic timed automata.

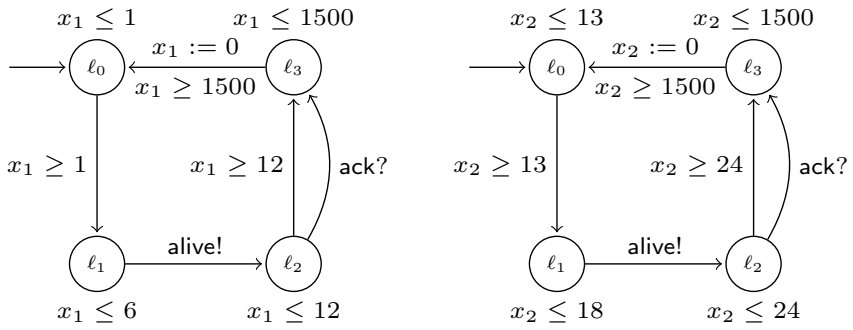


Figure 4.2: Two sensors as sequential timed automata. Left, sensor 1. Right sensor 2. The sensors are components of the fire alarm system presented in Section 1.2.

Theorem 4.2. *Let* $(L, \Sigma, \mathbb{X}, I, E, \ell_0, \ell_{\mathsf{fin}}, \mathsf{sta}, \mathsf{fin}, pt, e_{\mathsf{fin}}, \hat{x})$ *be a sequential timed automaton. Then* $(L, \Sigma, \mathbb{X}, I, E, \ell_0)$ *is periodic cyclic with period* pt*.*

Proof. Formulas saCyclic, saPeriod and the guard in $\hat{x} \geq p$ in e_{fin} ensure that there is an action transition from $\langle \ell_{\mathsf{fin}}, \nu \rangle, p \cdot p \xrightarrow{\alpha} \langle \ell_0, \nu' \rangle, p \cdot p$ for some $p \in \mathbb{N}$ for all computations of $\mathcal{A}$. Formulas saStart and saStartTime ensure that $\langle \ell_0, \nu' \rangle, p$ has a delay predecesor. Therefore, $\langle \ell_0, \nu' \rangle, p \in \mathsf{Start}(\mathcal{A})$. Formulas, saActive and saPeriod and the guard in e_{fin} ensure that $\langle \ell_{\mathsf{fin}}, \nu \rangle, p \cdot p$ has a delay predecesor, since they require $\mathsf{fin} < p$ and the guard in e_{fin} and the invariant ℓ_{fin} require $\hat{x} \leq p$ and $\hat{x} \geq p$. Formula saOneFin ensure the uniqueness of ℓ_{fin}. Thus $\mathcal{A}$ is periodic cyclic with period pt. □

Example 4.3. Consider the two sensors from Figure 4.2. The sensors are components from the fire alarm system presented in Section 1.2. Let $\mathcal{A}_1 = (L_1, \Sigma_1, \mathbb{X}_1, I_1, E_1, \ell_{1_0})$, $\mathcal{A}_2 = (L_2, \Sigma_2, \mathbb{X}_2, I_2, E_2, \ell_{2_0})$ denote the sensor 1 and the sensor 2. Clearly, $\mathcal{A}_1$ and $\mathcal{A}_2$ are timed automata. We will now represent the sensors as sequential timed automata. Let $\mathcal{A}'_1$ and $\mathcal{A}'_2$ denote the corresponding sequential timed automata for sensor 1 and sensor 2. Then,

$$\mathcal{A}'_1 = (L_1, \Sigma_1, \mathbb{X}_1, I_1, E_1, \ell_{1_0}, \ell_3, 1, 12, 1500, (\ell_3, \tau, x_1 \geq 1500, x_1, \ell_0), x_1)$$
$$\mathcal{A}'_2 = (L_2, \Sigma_2, \mathbb{X}_2, I_2, E_2, \ell_{2_0}, \ell_3, 13, 24, 1500, (\ell_3, \tau, x_2 \geq 1500, x_2, \ell_0), x_2)$$

Note that $\mathcal{A}'_1$ and $\mathcal{A}'_2$ satisfy the syntactic and semantic restrictions of sequential timed automata. Therefore, by Theorem 4.2 $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are periodic cyclic with period 1500.

♢

Before applying the sequential operator we must be sure that the activity phases of the automata are disjoint. One goal of sequential timed automata is to make the check of the sequentialisable assumptions from Definion 3.10 efficient. In sequential automata sequentialisability can be syntacticaly proven, as the following lemma shows.

Lemma 4.4. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequential timed automata with the same period pt and final time fin_1 of $\mathcal{A}_1$ strictly smaller than the start time sta_2 of $\mathcal{A}_2$, i.e. $\mathsf{fin}_1 < \mathsf{sta}_2$. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are sequentialisable.*

Proof. By Theorem 4.2, $\mathcal{A}_1$ and $\mathcal{A}_2$ are periodic cyclic with period pt. We now show, that for any $p \in \mathbb{N}$, the following holds

$$\sup(\mathsf{Active}(\mathcal{A}_1) \cap (pt \cdot i, pt \cdot (i+1))) < \inf(\mathsf{Active}(\mathcal{A}_2) \cap (pt \cdot i, pt \cdot (i+1)))$$

Formulas saActive, saStartTime and saFinalTime ensure that for time interval $(pt \cdot p, pt \cdot (p+1))$ the active points of $\mathcal{A}_1$ are in the interval $[(pt \cdot p) + \mathsf{sta}_1, (pt \cdot p) + \mathsf{fin}_1]$ and for $\mathcal{A}_2$ in the interval $[(pt \cdot p) + \mathsf{sta}_2, (pt \cdot p) + \mathsf{fin}_2]$. Therefore, $\sup(\mathsf{Active}(\mathcal{A}_1) \cap (pt \cdot p, pt \cdot (p+1))) = \sup([(pt \cdot p) + \mathsf{sta}_1, (pt \cdot p) + \mathsf{fin}_1]) = (pt \cdot p) + \mathsf{fin}_1$ and $\inf(\mathsf{Active}(\mathcal{A}_2) \cap (pt \cdot p, pt \cdot (p+1))) = \inf([(pt \cdot p) + \mathsf{sta}_2, (pt \cdot p) + \mathsf{fin}_2]) = (pt \cdot p) + \mathsf{sta}_2$. Since $\mathsf{fin}_1 < \mathsf{sta}_2$ it follows $(pt \cdot p) + \mathsf{fin}_1 < (pt \cdot p) + \mathsf{sta}_2$. □

Once the syntactic and semantic restrictions of sequential automata have been proven. Checking that these two sequential automata are sequentialisable, reduces to comparing their periods and their start and final points of activity. This check can be performed in constant time.

Example 4.5. Consider the two sensors from Figure 4.2. The sensors are components from the fire alarm system presented in Section 1.2. Now consider their corresponding sequential timed automata,

$$\mathcal{A}'_1 = (L_1, \Sigma_1, \mathbb{X}_1, I_1, E_1, \ell_{1_0}, \ell_3, 1, 12, 1500, (\ell_3, \tau, x_1 \geq 1500, x_1, \ell_0), x_1)$$
$$\mathcal{A}'_2 = (L_2, \Sigma_2, \mathbb{X}_2, I_2, E_2, \ell_{2_0}, \ell_3, 13, 24, 1500, (\ell_3, \tau, x_2 \geq 1500, x_2, \ell_0), x_2)$$

as defined in Example 4.3. In order to check if the two sequential automata are sequentialisable. Verify that their period is the same. This is the case since both have a period of 1500. Finaly check that $12 = \mathsf{fin}_1 < 13 = \mathsf{sta}_2$. Since, $\mathcal{A}'_1$ and $\mathcal{A}'_2$ satisfy the conditions from Lemma 4.4. $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are sequentialisable.

♢

4.2 Overclocks

The interleaving semantics of timed automata are often the source of state explosion problem. For sequentialisable timed automata, using the concatenation operator or the parallel composition operator, may cause the state space to grow expontentially in the number of components. Consider the parallel product or the concatenation of n sequential timed automata (see Figure 4.1) which are sequentialisable. The automaton will include a diamond like structure corresponding to the product of the n final locations. This structure will have 2^n locations and $n!$ paths. In this section, we present an important observation on the clock behaviour of TDMA based systems. This observation is formalized by the notion of overclocks. A reduction on clocks using overclocks is a solution to the state explosion problem generated by interleaving semantics of timed automata.

The principle behind the notion of overclocks is that TDMA components work synchronously, every componet using its own clock. However, from the observer point of view, one clock would suffice to measure the time and operations of the components. Intuitively, given two timed automata with one clock each. A clock is an overclock for the other two clocks, if for all computations the clocks are equal to the overclock, or they are at their reset configurations. That is, clocks are always equal or they are at their reset locations. For sequentialisable timed automata, clocks are different only at the reset points of time. We continue with the formal definition of overclock.

Definition 4.6 (Overclock)**.** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be timed automata with clocks x_1 and x_2, respectively. A clock $\hat{o}$ of $\mathcal{A}_1$ or $\mathcal{A}_2$ is an* overclock *for x_1 and x_2 in $\mathcal{A}_1 \| \mathcal{A}_2$ if and only if*

$$\forall \pi \in \mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2), j \in \mathbb{N}_0, \langle (\ell_1, \ell_2), \nu \rangle, t \in \mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2) \bullet \pi_j = \langle (\ell_1, \ell_2), \nu \rangle, t$$
$$\implies \nu \models (x_1 = \hat{o} \wedge x_2 = \hat{o}) \vee \ell_1 \in L_{\mathsf{rst}_1} \vee \ell_2 \in L_{\mathsf{rst}_2}.$$

♢

Example 4.7. Consider sensor 1 denoted by $\mathcal{A}_1$ and sensor 2 denoted by $\mathcal{A}_2$ from Figure 4.2. The sensors are components from the fire alarm system presented in Section 1.2. The corresponding sequential timed automata are given in Example 4.3. By Definition 3.1, the sets of reset configurations of $\mathcal{A}_1$ and $\mathcal{A}_2$ are $L_{\mathsf{rst}_1} = \{\ell_0, \ell_3\}$ and $L_{\mathsf{rst}_2} = \{\ell_0, \ell_3\}$ respectively. In the transistion system $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$, the set of reset locations L_{rst} is $\{(\ell_0, \ell_0), (\ell_0, \ell_3), (\ell_3, \ell_0), (\ell_3, \ell_3)\}$. Note, that for all computations in $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ the master clocks $\hat{x}_1$ and $\hat{x}_2$ are always equal (or equal to the overclock) except at some reset locations in L_{rst}. In particular, note that the clocks are different during intervals of zero time duration.

♢

If two clocks which are equal, except at certain configurations. An Overclock can be used to substitute the clocks and thus improve the verification time. The following lemma shows that for sequential timed automata with the same period, there is always an Overclock.

Lemma 4.8. *Given sequential timed automata, $\mathcal{A}_1, \mathcal{A}_2$ with period pt and masterclocks $\hat{x}_1$, $\hat{x}_2$ respectively. Then, there exist an overclock $\hat{o}$ for $\hat{x}_1$ and $\hat{x}_2$.*

Proof. Add a clock $\hat{o}$ to $\mathcal{A}_1||\mathcal{A}_2$ such that $\hat{o}$ is reset toguether with either $\hat{x}_1$ or $\hat{x}_2$. By the interleaving semantics of $||$ and since $\mathcal{A}_1$ and $\mathcal{A}_2$ are sequential automata in particular that they satisfy the following formulas: saOneReset and saOneFin. It follows that the resets can only occur at $\ell_1 \in L_{\mathsf{fin}_1}$ and $\ell_2 \in L_{\mathsf{fin}_2}$. By Theorem 4.2 and Theorem 3.7, $\mathcal{A}_1||\mathcal{A}_2$ is peridic cyclic with period pt. Therefore, this are the only reset points for all computations of $\mathcal{A}_1||\mathcal{A}_2$. □

4.3 Sequential Composition for Sequential Timed Automata

In Chapter 3, we introduced the concept of sequentialisable timed automata. We then introduced a concatenation operator on sequentialisable timed automata. This operator exploits the disjoint activity of sequentialisable timed automata. For sequentialisable timed automata, applying the concatenation operator yield a transition system which is bisimilar to the corresponding one obtained by parallel composition, with a reduced number of locations and edges. The use of the concatenation operator yield quadratic speed ups. Although this is an important result. The verification of a large number of sequentialisable timed automata can still be intractable. This is because, at the points of time which are multiples of the period, non-disjoint activity is allowed. That is, all automata might be active at some points of time. Because of the interleaving semantics, this fact can lead to a state explosion. The following example elucidates this problem.

Example 4.9. Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be sequentialisable sequential timed automata with period pt and masterclocks $\hat{x}_1, \ldots, \hat{x}_n$. This automata, have disjoint activity except at the points of time which are multiples of the period. i.e. time points $pt \cdot p$ for $p \in \mathbb{N}_+$.

Now consider the bisimilar systems $\mathcal{TS}(\mathcal{A}_1 \cdot \ldots \cdot \mathcal{A}_n)$ and $\mathcal{TS}(\mathcal{A}_1 \| \ldots \| \mathcal{A}_n)$. At the points of time where automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are active. Both the concatenation and the parallel composition operator, will construct the product of the locations involved in the activity. in this case the Cartesian product of the reset locations i.e. $L_{\mathsf{rst}_1} \times \cdots \times L_{\mathsf{rst}_n}$. The result is a diamond like structure with 2^n locations and $n!$ zero time interleaving sequences. In the case of the fire alarm system presented in Section 1.2, the diamond structure will have 2^{125} locations and at least 125! possible interleavings. The number

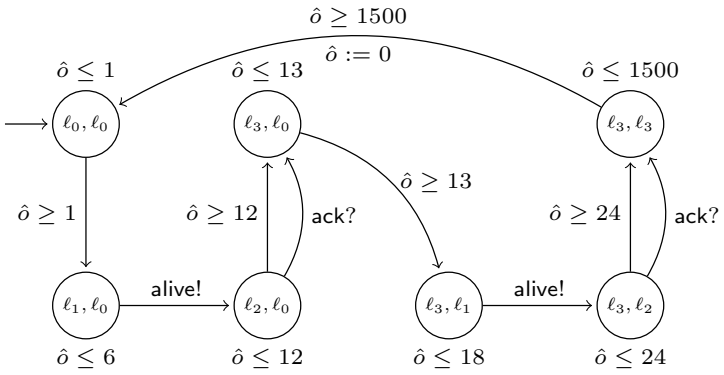


Figure 4.3: Sequential composition of sensors one and two

of locations and interleavings will render turn the verification task to intractable. In addition, note that clocks $\hat{x}_1, \ldots, \hat{x}_n$ are always equal up to the time points $pt \cdot p$ for $p \in \mathbb{N}_+$.

By Lemma 4.8, there exist an overclock which is equal to one or both of these clocks at all points of time except at the multiples of the period. This suggest a clock simplification.

$\diamondsuit$

Fortunately, for sequentialisable sequential timed automata. We can easily identify the points of time with non-disjoint activity. Further, at this points of time the allowed actions are restricted. Thus, we can tackle the state explosion problem at this points of time while preserving most of the properties.

Our approach consist in defining a sequential composition operator on sequentialisable sequential timed automata. This operator exploits the syntactic properties of sequential timed automata and further improves the concatenation operator. The sequential composition operator eliminates complete the interleavings and summarize the actions at the non-disjoint activity points of time.

Intuitively, for two sequentialisable sequential timed automata, the sequential composition operator works as follows. First, the first automaton is active and performs his actions while the second automaton is at his initial location. Then, the second automaton is active and performs his actions while the first automaton is at his final location. Next, both automata are active and at their final locations. We know that there is only one edge for automaton that is enabled at these points of time. This edge resets the corresponding master clock. The sequential composition operator summarizes the effect of the resets by replacing the master clocks by one overclock and do not introduce the locations resulting from the product of the reset locations. We continue with the formal definition of the sequential composition operator.

Definition 4.10 (Sequential Composition)**.** *Let* $\mathcal{A}_i$*,*

$$\mathcal{A}_i = (L_i, \Sigma_i, \mathbb{X}_i, I_i, E_i, \ell_{0_i}, \ell_{\mathsf{fin}_i}, \mathsf{sta}_i, \mathsf{fin}_i, pt, e_{\mathsf{fin}_i}, \hat{x}_i), i = 1, 2,$$

be sequential timed automata.

Then the sequential composition *of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *yields the tuple*

$$\mathcal{A}_1 \fatsemi \mathcal{A}_2 \stackrel{\mathsf{def}}{=} (L, \Sigma_1 \cup \Sigma_2, \mathbb{X}_1 \cup \mathbb{X}_2, I, E, (\ell_{0_1}, \ell_{0_2}), (\ell_{\mathsf{fin}_1}, \ell_{\mathsf{fin}_2}), \mathsf{sta}_1, \mathsf{fin}_2, pt, e_{\mathsf{fin}}, \hat{o})$$

where

- *the master clock* $\hat{o}$ *is an overclock for* $\hat{x}_1$*,* $\hat{x}_2$*.*
- *the set of locations consists of pairs where* $\mathcal{A}_2$ *assumes an initial or* $\mathcal{A}_1$ *assumes a final location, i.e.*

  $$L = ((L_1 \setminus \{\ell_{\mathsf{fin}_1}\}) \times \{\ell_{0_2}\}) \cup (\{\ell_{\mathsf{fin}_1}\} \times L_2),$$

- *the final edge* e_{fin} *is*

  $$((\ell_{\mathsf{fin}_1}, \ell_{\mathsf{fin}_2}), \varnothing, \varphi_1 \wedge \varphi_2, Y_1 \cup Y_2, (\ell_{0_1}, \ell_{0_2}))$$

  given the final edges $e_{\mathsf{fin}_i} = (\ell_{\mathsf{fin}_i}, \varnothing, \varphi_i, Y_i, \ell_{0_i})$*,* $i = 1, 2$*,*

- *the clock constraint of location* (ℓ_1, ℓ_2) *is the conjunction of the corresponding clock constraints in* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *where substitute each* $\hat{x}_1$ *and* $\hat{x}_2$ *is syntactically substituted by* $\hat{o}$*, i.e.*

  $$I(\ell_1, \ell_2) = (I_1(\ell_1) \wedge I(\ell_2))[\hat{x}_1/\hat{o}, \hat{x}_2/\hat{o}],$$

- *the set of edges comprises* e_{fin} *and compositions of* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *edges where* $\hat{x}_1$ *and* $\hat{x}_2$ *are substituted by* $\hat{x}$ *in guards and reset sets, i.e.*

  $$\begin{aligned} E = \{e_{\mathsf{fin}}\} &\cup \{((\ell_1, \ell_{0_2}), \alpha, \tilde{\varphi}_1, \tilde{Y}_1, (\ell_1', \ell_{0_2})) \mid (\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1 \setminus \{e_{\mathsf{fin}_1}\}\} \\ &\cup \{((\ell_{\mathsf{fin}_1}, \ell_2), \alpha, \tilde{\varphi}_2, \tilde{Y}_2, (\ell_{\mathsf{fin}_1}, \ell_2')) \mid (\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2 \setminus \{e_{\mathsf{fin}_2}\}\} \end{aligned}$$

  where $\tilde{\varphi}_i = \varphi_i[\hat{x}_1/\hat{o}, \hat{x}_2/\hat{o}]$*,* $i = 1, 2$*, and* $\tilde{Y}_i = Y_i[\hat{x}_1/\hat{o}, \hat{x}_2/\hat{o}]$*,* $i = 1, 2$*.*

♢

Given two sequentialisable sequential timed automata, the sequential composition operator yields a sequential timed automaton. The automaton has the start point of its activity from the start point of activity of the first automata. The automaton has the final point of its activity from the final point of activity of the second automata. The final edge is a combination of the final edges of the corresponding automata. The initial and final locations are pairs of the initial and final locations of the corresponding automata. Note, that the master clocks are been substituted in all the invariants and edges by an overclock.

Example 4.11. Consider the two sensors from Figure 4.2. The corresponding sequential timed automata are described in Example 4.3. The resulting sequential timed automata of applying the sequential composition operator on these automata is depicted in Figure 4.3 After the activity of sensor 1 is done, sensor 1 is at his final location and there is an edge going to the start location of sensor 2. At point of time 1500 both sensors are active and both move simultaneously to their initial locations. Note that the master clocks, $\hat{x}_1$ and $\hat{x}_2$ are replaced by the overclock $\hat{o}$.

Consider Figure 3.3 corresponding to the reachable automaton obtained by parallel composition of sensor 1 and sensor 2. Note that the diamond structure corresponding to the product of the reset locations is not present. For the fire alarm system presented in Section 1.2, the diamond structure with 2^{125} locations and 125! paths is not present. In Section 6 we present our experimental results. We will show that the verification time for the fire alarm system, goes from exponential to linear. ♢

In Section 3.3, we have shown that for sequentialisable sequential automata the transitions systems corresponding to their parallel composition and their concatenation are bisimilar. In what follows, we will show that the transition system obtained by their sequential composition is weak bisimilar. The weak bisimilarity reflects the effect of removing the diamond like structure generated by the Cartesian product of reset locations. In particular, the configurations occurring at reset points of time, have no delay successors but action successors. We use the following definition in order to simplify a definition of weak bisimulation.

Definition 4.12 (Action Reachability)**.** *Let $\mathcal{A}$ be a timed automaton and $c, c' \in Conf(\mathcal{A})$ configurations. We say c' is* reachable *in $\mathcal{A}$ from c* via action transitions*, denoted by* $\mathsf{actReach}(c, c', \mathcal{A})$*, if and only if*

$$\mathsf{actReach}(c, c', \mathcal{A}) \overset{\mathsf{def}}{\Longleftrightarrow} \exists c_0, \ldots, c_n \in Conf(\mathcal{A}) \bullet c_0 = c \wedge c_n = c' \\ \wedge \forall 0 \leq j < n \in \mathbb{N}_0 \bullet c_j \xrightarrow{\lambda_j} c_{j+1} \wedge \lambda_j \in \Sigma.$$

♢

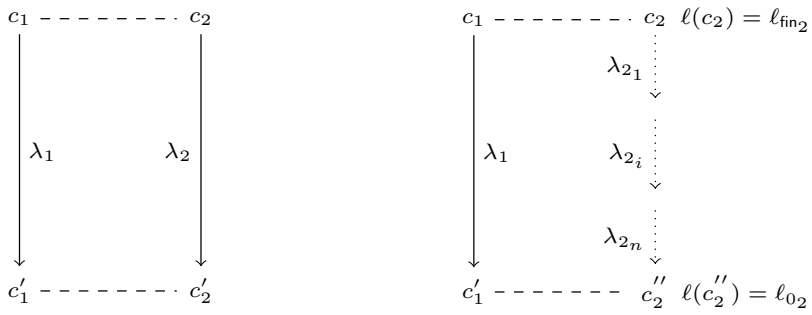


Figure 4.4: Weak bisimulation relation for timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$. Timed automaton $\mathcal{A}_1$ is the result of applying the sequential composition operator whereas $\mathcal{A}_2$ is the result of applying the parallel composition operator. Left, weak bisimulation relation in points of time different than multiples of the period. Right, weak bisimulation relation at the period points of time, note that $\mathcal{A}_2$ performs a number of action transitions.

Definition 4.13 (Weak bisimulation)**.** *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequential automata with $\hat{o} \in \mathbb{X}_1$ and $\hat{x}_1, \hat{x}_2 \in \mathbb{X}_2$ such that $\hat{o}$ is an overclock for $\hat{x}_1, \hat{x}_2$ and let*

$$\mathcal{TS}_i(\mathcal{A}_i) = (\mathit{Conf}(\mathcal{A}_i), \mathbb{R} \cup \Sigma_i, \{\xrightarrow{\lambda^i} | \ \lambda^i \in \mathbb{R} \cup \Sigma_i\}, C_{0_i}), i = 1, 2,$$

be the corresponding labelled transition systems.

A relation $\mathcal{W} \subseteq \mathit{Conf}(\mathcal{A}_1) \times \mathit{Conf}(\mathcal{A}_2)$ is called weak bisimulation *of $\mathcal{A}_1$ and $\mathcal{A}_2$ if and only if it satisfies the following conditions.*

1. $\forall c_1 \in C_{0_1} \ \exists c_2 \in C_{0_2} \bullet (c_1, c_2) \in \mathcal{W}$ *and* $\forall c_2 \in C_{0_2} \ \exists c_1 \in C_{0_1} \bullet (c_1, c_2) \in \mathcal{W}$
2. *for all* $(c_1 = (\langle \ell_1, \nu_1 \rangle, t_1), c_2 = (\langle \ell_2, \nu_2 \rangle, t_2)) \in \mathcal{W}$,
   a) $\beta(\nu_1) = \nu_2$, $t_1 = t_2$ *with* $\beta(\nu) \stackrel{\mathsf{def}}{=} \nu|_{\hat{o}} \cup \{\nu_{\hat{x}_1} \mapsto \nu(\hat{o}), \nu_{\hat{x}_2} \mapsto \nu(\hat{o})\}$,
   b) $\forall c_1 \xrightarrow{\lambda^1} c_1' \bullet (\exists c_2 \xrightarrow{\lambda^2} c_2' \bullet (c_1', c_2') \in \mathcal{W}) \vee (\ell(c_1) = \ell_{\mathsf{fin}_1} \wedge$
   $\exists c_2'' \bullet \mathsf{actReach}(c_2, c_2'', \mathcal{A}_2) \wedge \ell(c_2'') = \ell_{0_2} \wedge (c_1', c_2'') \in \mathcal{W}),$
   c) $\forall c_2 \xrightarrow{\lambda^2} c_2' \ \exists c_1 \xrightarrow{\lambda^1} c_1' \bullet (c_1', c_2') \in \mathcal{W} \vee (\ell(c_2) = \ell_{\mathsf{fin}_2} \wedge$
   $\exists c_2'' \bullet \mathsf{actReach}(c_2, c_2'', \mathcal{A}_2) \wedge \ell(c_2'') = \ell_{0_2} \wedge (c_1', c_2'') \in \mathcal{W}),$

$\mathcal{A}_1$ is called weakly bisimilar to $\mathcal{A}_2$ iff there is a weak bisimulation of $\mathcal{A}_1$ and $\mathcal{A}_2$.

$\diamondsuit$

Our definition of weak bisimulation is a specific one characterizing the differences between the system obtained by parallel composition and the one obtained by sequential composition. Note, that a points of time which are different than the multiples of the period, both systems perform the same number of steps. The difference and the reason why the relation is a weak bisimulation and not a simulation relation, is that at points of time which are multiples of the period, there are a number of action interleavings in the system resulting from application of parallel composition. Whereas, the system resulting from application of the sequential composition operator performs only one step. This is illustrated in Figure 4.4 right.

The following theorem, states that for sequentialisable sequential timed automata. The transition system induced by the automaton obtained by application of the sequential composition operator is weak-bisimilar with respect to the one obtained by application of the parallel composition operator.

Theorem 4.14. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequential timed automata with the same period pt and final time fin_1 of $\mathcal{A}_1$ strictly smaller than the start time sta_2 of $\mathcal{A}_2$, i.e. $\mathsf{fin}_1 < \mathsf{sta}_2$. Then $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is weak-bisimilar to $\mathcal{A}_1 \| \mathcal{A}_2$.*

Proof. For the proof let $\hat{o}$ be the overclock of $\mathcal{A}_1 \fatsemi \mathcal{A}_2$, i.e. $\hat{o} \in \mathbb{X}_1$, let the masterclocks $\hat{x}_1, \hat{x}_2$ be the master clocks in $\mathcal{A}_1 \| \mathcal{A}_2$, i.e. $\hat{x}_1, \hat{x}_2 \in \mathbb{X}_2$, let configuration $c_1 = (\langle \ell, \nu \rangle, t) \in \mathit{Conf}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$, and configuration $c_2 = (\langle \ell, \nu \rangle, t) \in \mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2)$. In the proof we will find a correspondence between configurations in $\mathit{Conf}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$ and configurations in $\mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2)$. For this we will use functions Γ and β on configurations and valuations respectively. Note that the definition of the function β is given in the definition of weak bisimulation. The function β takes valuations where the overclock is present and uses the value of the overclock to restore the value of the corresponding masterclocks. The function Γ is a function which applies β to configurations in $\mathit{Conf}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$, i.e.

$$\Gamma(\langle \ell, \nu \rangle, t) \stackrel{\mathsf{def}}{=} \langle \ell, \beta(\nu) \rangle, t$$

The function β^{-1} be a function which substitutes clocks $\hat{x}_1$ and $\hat{x}_2$ by $\hat{o}$. i.e.

$$\beta^{-1}(\nu) \stackrel{\mathsf{def}}{=} \nu|_{\hat{x}_1, \hat{x}_2} \cup \{\nu(\hat{o}) \mapsto \nu(\hat{x}_1)\}$$

and Γ^{-1} the function which applied β^{-1} to configurations in $\mathit{Conf}(\mathcal{A}_1 \| \mathcal{A}_2)$.

$$\Gamma^{-1}(\langle \ell, \nu \rangle, t) \stackrel{\mathsf{def}}{=} \langle \ell, \beta^{-1}(\nu) \rangle, t$$

Now construct the weak bisimulation relation $\mathcal{W}$ as follows,

$$\mathcal{W} = \{(c, \Gamma(c)) \mid c \in \mathit{Conf}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)\}.$$

For condition (1) we have that $c_{0_1} = \langle \ell_{0_1}, \nu_0 \rangle, 0$ and $\Gamma(c_{0_1}) = c_{0_2}$. Therefore, $(c_{0_1}, c_{0_2}) \in \mathcal{W}$ and the unique initial locations are in relation.
For condition (2), let $(c_1, c_2) \in \mathcal{W}$. By definition of $\mathcal{W}$ we have that $c_2 = \Gamma(c_1)$. Now consider the following cases:

1. Condition (2.a). By construction of $\mathcal{W}$ we have that $c_1 = \langle \ell_1, \nu_1 \rangle, t_1$ and $c_2 = \langle \ell_1, \beta(\nu_1) \rangle, t_1$. Therefore $\beta(\nu_1) = \nu_2$ and $t_1 = t_2$.

2. Condition (2.b). There is either a delay or an action transition in $\mathcal{A}_1 \fatsemi \mathcal{A}_2$.

   - Delay Transitions: there is a transition,

     $$c_1 = \langle (\ell_1, \ell_2), \nu_1 \rangle, t_1 \xrightarrow{t'} \langle (\ell_1, \ell_2), \nu_1 + t' \rangle, t_1 + t' = c_1'$$

     such that $\nu_1 + t'' \models I_1(\ell_1) \wedge I_2(\ell_2)$ for all $t'' \in [0, t']$. Now by definition of overclock $\beta(\nu_1) + t'' \models \beta(I_1(\ell_1)) \wedge \beta(I_2(\ell_2))$ for all $t'' \in [0, t']$. Now let $c_2' = \langle (\ell_1, \ell_2), \beta(\nu_1) + t' \rangle, t_1 + t'$. Then $c_2 \xrightarrow{t'} c_2'$ and $(c_1', c_2') \in \mathcal{W}$ since $c_2' = c_1'$.

   - Action Transitions: we consider two cases first the case when $\mathcal{A}_1$ is at is final location i.e. $\ell(c_1) = \ell_{\mathsf{fin}_1}$ and second the case when $\mathcal{A}_1$ is not at is final location i.e. $\ell(c_1) \neq \ell_{\mathsf{fin}_1}$.

     – $\ell(c_1) = \ell_{\mathsf{fin}_1}$. Intuition, $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ takes the edge e_{fin_1} from ℓ_{fin_1} to ℓ_{0_1}, this edge summarizes the diamond generated by the interleaving semantics of the parallel product operator, the diamond involves the locations in the set L_{fin} of $\mathcal{A}_1 || \mathcal{A}_2$, by Theorem 3.7, $\mathcal{A}_1 || \mathcal{A}_2$ is periodic cyclic, therefore it has to reach ℓ_{fin_2}. Formally: We have $c_1 = \langle \ell_{\mathsf{fin}_1}, \nu_1 \rangle, t$ and $c_2 = \langle \ell_{\mathsf{fin}_1}, \beta(\nu_1) \rangle, t$. By definition of e_{fin_1}, $c_1 = \langle \ell_{\mathsf{fin}_1}, \nu_1 \rangle, t_1 \xrightarrow{\alpha} \langle \ell_{0_1}, \nu_1[\{\hat{o}\} := 0] \rangle, t_1 = c_1'$. Since $\mathcal{A}_1 || \mathcal{A}_2$ is periodic cyclic there exist an action path of the form $c_2 \xrightarrow{Y_1} c_2' \xrightarrow{Y_2} c_2''$ and $c_2'' = \langle \ell_{0_2}, \nu_1[Y_1 \cup Y_2 := 0] \rangle, t_1$ and $\Gamma(c_1') = c_2''$. Therefore, $(c_1', c_2'') \in \mathcal{W}$.

     – $\ell(c_1) \neq \ell_{\mathsf{fin}_1}$. There is an action transition

       $$c_1 = \langle (\ell_1, \ell_2), \nu_1 \rangle, t_1 \xrightarrow{\alpha} \langle (\ell_1', \ell_2), \nu_1' \rangle, t_1 = c_1'$$

       then there is an edge $e = (\ell_1, \alpha, \varphi, Y, \ell_1') \in E_1$ such that $\nu_1 \models \varphi$, $\nu_1' = \nu_1[Y := 0]$ and $\nu_1' \models I_1(\ell_1)$. By definition of parallel composition operator $\|$ such an edge exists in $\mathcal{A}_1 \| \mathcal{A}_2$, and since $\hat{o}$ is an overclock for $\hat{x}_1, \hat{x}_2$,

it follows that $\beta(\nu_1) \models (\varphi[\hat{o}/\hat{x}_1])$ and $\beta(\nu_1') = \beta(v_1)[Y := 0]$. Now let $c_2 \xrightarrow{\alpha} c_2'$ be the action transition induced by e with $c_2' = \langle(\ell_1', \ell_2), \nu_1'\rangle, t_1$. Clearly $(c_1', c_2') \in \mathcal{W}$, since $\Gamma(c_1') = c_2'$.

3. Condition (2.c). There is either a delay or an action transition in $\mathcal{A}_1 \| \mathcal{A}_2$.

   - Delay transitions: there is a transition,

   $$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t_2 \xrightarrow{t'} \langle(\ell_1, \ell_2), \nu_2 + t'\rangle, t_2 + t' = c_2'$$

   such that $\nu_2 + t'' \models I_1(\ell_1) \wedge I_2(\ell_2)$ for all $t'' \in [0, t']$. Since $\Gamma(c_1) = c_2$ we have that $\beta(\nu_1) = \nu_2$ and $t_1 = t_2$ Therefore, $\nu_1 + t'' \models I_1(\ell_1) \wedge I_2(\ell_2)$ and $c_1' = \langle(\ell_1, \ell_2), \nu_1 + t''\rangle, t_1 + t'' \in \mathit{Conf}(\mathcal{A}_1 ; \mathcal{A}_2)$. Which implies that $(c_1', c_2') \in \mathcal{W}$, since $\Gamma(c_1') = c_2'$.

   - Action transitions: we need to distinguish three cases. First, when $\mathcal{A}_1$ is active, i.e. the points in $\mathsf{Active}(\mathcal{A}_1)$. Second, when $\mathcal{A}_2$ is active, i.e. the points in $\mathsf{Active}(\mathcal{A}_2)$. Finally, when both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active. i.e. at points of time $pt \cdot p$ for some $p \in \mathbb{N}$. Note that Theorem 3.7 ensures that $\ell(c_2) = \ell_{\mathsf{fin}_2}$ is only possible if c_2 has time stamp $pt \cdot p$ for some $p \in \mathbb{N}$.

     – Time points in $\mathsf{Active}(\mathcal{A}_1)$. Let $t \in \mathsf{Active}(\mathcal{A}_1)$. There are action transitions in $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ corresponding to some edge in E_1. Thus we have

     $$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t \xrightarrow{\alpha} \langle(\ell_1', \ell_2), \nu_2'\rangle, t = c_2'$$

     then by definition of parallel product and the fact that $t \in \mathsf{Active}(\mathcal{A}_1)$ we know that there exist an edge $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$ such that $\nu_2 \models \varphi_1$, $\nu_2' = \nu_2[Y := 0]$ and $\nu_2' \models I_1(\ell_1') \wedge I_2(\ell_2)$. By lemma 3.13 we know that $\ell_2 = \ell_{0_2}$ and by definition of the sequential composition operator, we know that there exist an edge $e = ((\ell_1, \ell_2), \alpha, \varphi_1, Y_1, (\ell_1', \ell_2))$ in $\mathcal{A}_1 ; \mathcal{A}_2$ such that $(\ell_1, \alpha, \varphi_1, Y_1, \ell_1') \in E_1$. Now by using edge e complete $c_1 = \Gamma^{-1}(c_2) \xrightarrow{\alpha} c_1'$ with $c_1' = \langle(\ell_1', \ell_2), \nu_1'\rangle, t_1'$ where $\nu_1' = \beta^{-1}(\nu_2')$ and $\nu_2' = \nu_2[Y := 0]$. It follows that $\nu_1' \models I_{1_1}(\ell_1') \wedge I_{1_2}(\ell_2)$. Therefore, $c_1' \in \mathit{Conf}(\mathcal{A}_1 ; \mathcal{A}_2)$, $c_1' = \Gamma^{-1}(c_2')$ and $(c_1', c_2') \in \mathcal{W}$.

     – Time points in $\mathsf{Active}(\mathcal{A}_2)$: there are action transitions in $\mathcal{TS}(\mathcal{A}_1 || \mathcal{A}_2)$ corresponding to some edge in E_2. Thus we have

     $$c_2 = \langle(\ell_1, \ell_2), \nu_2\rangle, t \xrightarrow{\alpha} \langle(\ell_1, \ell_2'), \nu_2'\rangle, t = c_2'$$

then by definition of parallel product and the fact that $t \in \mathsf{Active}(\mathcal{A}_2)$ we know that there exist an edge $(\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2$ such that $\nu_2 \models \varphi$, $\nu_2' = \nu_2[Y_2 := 0]$ and $\nu_2' \models I_1(\ell_1) \wedge I_2(\ell_2')$. By Lemma 3.13 we know that $\ell_1 = \ell_{\mathsf{fin}_1}$ and by definition of sequential composition we know that there exist an edge $e = ((\ell_1, \ell_2), \alpha, \varphi_2, Y_2, (\ell_1, \ell_2'))$ in $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ such that $(\ell_2, \alpha, \varphi_2, Y_2, \ell_2') \in E_2$ and $\ell_1 = \ell_{\mathsf{fin}_1}$. Now complete $\Gamma(c_1) = c_2 \xrightarrow{\alpha} c_1'$ with $c_1' = \Gamma^{-1}(c_2') = \langle (\ell_1, \ell_2'), \beta^{-1}(\nu_2') \rangle, t_1'$ with $\nu_1' = \beta^{-1}(\nu_2') = \beta(\nu_2[Y_2 := 0])$. It follows that $\beta(\nu_1') \models I_1(\ell_1) \wedge I_2(\ell_2')$. Therefore, $c_1' \in \mathit{Conf}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$ and $(c_1', c_2') \in \mathcal{W}$.

– Points of time $pt \cdot p$ for some $p \in \mathbb{N}_0$. Then, both $\mathcal{A}_1$ and $\mathcal{A}_2$ are active. Therefore, there are action transitions in $\mathcal{A}_1$ and $\mathcal{A}_2$. The reachable configurations of $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ with time stamp $pt \cdot p$ for some $p \in \mathbb{N}$ are either at locations ℓ_{fin} or ℓ_0. By the above fact and by construction of $\mathcal{W}$, c_2 can either be at location ℓ_{fin} or ℓ_0.

  Consider $\ell(c_2) = \ell_0$. Since $\mathcal{A}_1 \| \mathcal{A}_2$ is periodic cyclic, by definition the only possible transitions at time stamp $pt \cdot p$ are delay transitions. Above we have already considered delay transitions.

  Now, consider $\ell(c_2) = \ell_{\mathsf{fin}}$. Then there is a transition $c_2 \xrightarrow{\lambda^2} c_2'$By Theorem 3.7, $\mathcal{A}_1 \| \mathcal{A}_2$ is periodic cyclic and by lemma 3.13 we know that $\ell(c_2') \in L_{\mathsf{fin}_2}$ Then it follows that c_2 can reach via action transitions a configuration c_2'' such that $c_2 \xrightarrow{Y_1} c_2' \xrightarrow{Y_2} c_2''$ and $c_2'' = \langle \ell_{0_2}, (\nu_2)[Y_1 \cup Y_2 := 0] \rangle, t_2$. By definition of final edge, either $\mathcal{A}_1$ or $\mathcal{A}_2$ have performed the action transition induced by edge e_{fin_1} or e_{fin_2}. Let φ denote the guard of the corresponding edge. Then we have that either $\hat{x}_1 \models \varphi$ or $\hat{x}_2 \models \varphi$. Where by definition of final edge φ is of the form $x \geq pt$ for some clock x. Since $\hat{o}$ is an overclock for $\hat{x}_1$ and $\hat{x}_2$ we have that $\hat{o} \models \varphi$. Then, by definition of final edge we have $\hat{o} \geq pt$. Since e_{fin} is the unique edge from ℓ_{fin_1} we have e_{fin} induces the transition $c_1 \xrightarrow{\lambda^1} c_1'$ where $c_1' = \langle \ell_{0_1}, \nu_1[\{\hat{o}\} := 0] \rangle, t_2$ and $\Gamma(c_1') = c_2''$. Therefore, $(c_1', c_2'') \in \mathcal{W}$. This concludes our proof.

□

Since TDMA based systems include a large number of components. It is useful to describe the system in a compositional manner. For this it is important that the class of sequential timed automata is closed under the application of the sequential composition

operator. The following theorem states that when the sequential composition operator is applied to two sequentialisable sequential timed automata, the resulting automata is a sequential timed automata.

Theorem 4.15. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequential timed automata with the same period pt and final time fin_1 of $\mathcal{A}_1$ strictly smaller than the start time sta_2 of $\mathcal{A}_2$, i.e. $\mathsf{fin}_1 < \mathsf{sta}_2$. Then $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is a sequential timed automata with period pt.*

Proof. We need to show that the resulting timed automaton is a sequential timed automaton. i.e. it satisfies the constraints from Definition 4.1 of sequential timed automata. By definition of sequential composition the masterclocks of both automata are substituted by an overclock $\hat{o}$. Therefore, by construction, the syntactic constraints $\mathsf{saStart}, \mathsf{saStartTime}, \mathsf{saFinalTime}$ and $\mathsf{saPeriod}$ are satisfied. By assumption we have $\mathsf{fin}_1 < \mathsf{sta}_2$, since both automata have the same period *pt*, the condition $\mathsf{saActive}$ is satisfied. For Conditions $\mathsf{saOneReset}$ and $\mathsf{saOneFin}$, both automata had only one edge. The sequential operator combines these two edges and creates a new final edge, which satisfies these conditions. For the semantic constraint $\mathsf{saCyclic}$. First we use Theorem 3.7, which states that $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ is periodic cyclic. Then by Theorem 4.14, the transition system $\mathcal{TS}(\mathcal{A}_1 \| \mathcal{A}_2)$ is weak-bisimilar with respect to the transition system $\mathcal{TS}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$. This fact implies that the initial location (start configurations) of $\mathcal{TS}(\mathcal{A}_1 \fatsemi \mathcal{A}_2)$ are infinitely often visited. Since the initial location is the destination of the final location. The final location is infinitely often visited. Therefore, condition $\mathsf{saCyclic}$ is satisfied. Since all the conditions are satisfied, we conclude that $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is a sequential timed automata with period *pt*. □

A consequence of the above theorem is that for sequentialisable sequential timed automata. The application of the sequential composition operator, produces a periodic cyclic timed automaton.

Corollary 4.16. *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be sequential timed automata with the same period pt and final time fin_1 of $\mathcal{A}_1$ strictly smaller than the start time sta_2 of $\mathcal{A}_2$, i.e. $\mathsf{fin}_1 < \mathsf{sta}_2$. Then $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is periodic cyclic with period pt.*

Proof. By Theorem 4.15, we know that $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is a sequential timed automata with period *pt*. For sequential timed automata, Theorem 4.2, ensures the automata to be periodic cyclic. Therefore, $\mathcal{A}_1 \fatsemi \mathcal{A}_2$ is periodic cyclic with period *pt*. □

4.3.1 Case Study: Steer-By-Wire Architecture Using TTP/C

In order to illustrate the applicability of our method, we propose the intuition of a model for a system based on the Time-Triggered Architecture based on [62].

In most systems based on the Time-Triggered Architecture every node consist of a local CPU, a Communication Network Interface, and a TTP/C controller. The data communication over TTP/C is organized in TDMA rounds. A TDMA round is divided into slots and every node is assigned to a slot. A recurring sequence of TDMA rounds constitutes a cluster. The system can be globally monitored based on bus tracing.

A steer-by-wire system would require from 8 to 30 nodes. Interesting properties to verify might include: If a node fails, is this node detected as malfunctioning in within a TDMA round; or if a node fails, does the system still satisfy a given property.

Let us consider that the system has n nodes and one global monitor. The monitor could be modeled by one timed automaton $\mathcal{A}_m$. Every node could be modeled by Two timed Automata; one for the CPU $\mathcal{A}_{CPU}$ and one for the TTP/C communication $\mathcal{A}_{TTP}$. The automata corresponding to a node could communicate via shared variables.

A TDMA round would have the following form, $\mathcal{A}_m||\mathcal{A}_{CPU_1}||\ldots||\mathcal{A}_{CPU_n}||\mathcal{A}_{TTP_1}||\ldots||\mathcal{A}_{TTP_n}$. Since Automaton $\mathcal{A}_{TPP_i}$ will only perform actions on its corresponding slot for $i \in \{1,\ldots,n\}$, we can apply our method and obtain: $\mathcal{A}_m||\mathcal{A}_{CPU_1}||\ldots||\mathcal{A}_{CPU_n}||(\mathcal{A}_{TTP_1} \fatsemi \ldots \fatsemi \mathcal{A}_{TTP_n})$. Which we believe, would lead to an improvement on the verification time. Given a fixed number of TDMA rounds we can use the above method to construct a cluster.

4.4 Related Work

In [70] patterns are used in the context of timed systems. Patterns are used to specify timed properties. The use of patters allows non-experts to specify relevant system properties. This work is similar in spirit to ours, since we provide patterns to ensure relevant semantic properties of periodic cyclic timed automata. Further, the work in [70] focuses on system properties. We on the other side, focus on the model of the system.

Let us note that the technique of clock reduction of [54] and its generalization in [16], which may seem relevant in this context, are orthogonal to our approach. This is because, even if a successful clock reduction has taken place, the optimized model can profit from the disjoint activity, i.e. the model can be further optimized by reducing a number of unnecessary edges.

We have seen that the automaton obtained by parallel product of n sequential timed automata, at points of time which are multiples of the period, will have a diamond structure with 2^n locations and $n!$ paths. This diamond will hinder the verification procedure. Successful application of techniques such as partial order reduction [25, 52, 98, 72] or for the method proposed in [119] for dealing with interleavings, will remove the diamond structure. Note, that our approach not only removes the diamond as well, but also removes a number of unnecessary edges. The reduction of unnecessary edges lead to quadratic speed ups.

In Section 4.2 we introduce the notion of overclocks. A similar notion has been published in [76, 77], where quasi-equal clocks are introduced. In [101] an efficient method for detecting quasi-equal clocks is presented. After quasi-equal clocks have been detected, a clock simplification [76] can be performed. For the case of the fire alarm system, the two master clocks are quasi-equal and they are simplified and replaced by an overclock. In our experimental section, we show that the sequential composition operator, can further improve the clock optimization from [76] by removing unnecessary or unreachable edges.

In [106] a sequential composition operator is presented. However, the corresponding timed automata do not allow cyclic behavior as we do. Another important difference is that their approach is action oriented i.e. it exploits the independence of actions. In contrast our approach which is time oriented.

Chapter 5

Clock Optimizations for Timed Systems

Contents

Timed automata and timed model checking [5, 91, 140] have been successfully applied for the verification of real-time systems. Still, the number of clocks in a timed automaton will always be an issue for scalability and the optimization of timed model checking will always be a topic of research. Optimization techniques for timed model checking are often based on some notion of redundancy in the representation of the timed model and its behavior in terms of clock valuations. The detection of the corresponding redundancies is then a prerequisite for applying the optimization.

In the previous chapter, we have introduced a clock optimization method for sequentialisable sequential timed automata. The optimization consisted in replacing two master clocks by an overclock. The behavior of these two master clocks has been formalized under the notion of Quasi-equal clocks. Two clocks x and y in a given timed automaton are quasi-equal if the invariant $x = y \vee x = 0 \vee y = 0$ holds. That is, in every step in every transition sequence, the two clocks x and y have equal value except for steps where one of them has been reset but not the other. As a consequence, the invariant $x = y$ can be violated only at single time points (i.e. during time periods of length zero). In a way, the violation of the invariant is an artefact of the model of the behavior of a timed systems by discrete sequences.

Note that quasi-equal clocks are not restricted to TDMA based systems. In [76] an optimization using quasi-equal clocks is presented. In [76] the clock reduction method

starts with a given specification of *quasi-equal* clocks. In principle, the zone graph can be used to detect which clocks are quasi-equal; the construction of the zone graph would, however, defeat its very purpose (which is the optimization of this construction). In this chapter, we present an abstraction that is coarse enough to yield a drastic reduction of the zone graph and precise enough to identify a large class of quasi-clocks.

The abstraction is motivated by an intuition about the way quasi-equalities can be tracked. The intuition is that the behavior of a timed automaton over a non-zero period of time (without resets) will neither introduce new quasi-equalities nor "destroy" a quasi-equality and hence we can apply the most coarse abstraction there. In contrast, when different values for quasi-equal clocks arise in a sequence of configurations where time does not elapse, we must track the constants for the values of the clocks as precisely as possible (i.e. apply no proper abstraction). Thus, as an intermediate step for computing abstract zones, we must apply logical reasoning in order to infer whether the zone accounts for behavior of zero (as opposed to: non-zero) periods of time.

Our abstraction methods amounts to computing an abstraction of the zone graph. We use the abstract zone graph to detect quasi-equal clocks.

We have implemented our method in our tool sAsEt, which is built in top of the Jahob verification framework. This allows us to represent zones (and abstract zones) by linear real arithmetic formulas and to perform the required logical reasoning (on the duration of the corresponding period of time) through calls of an SMT solver.

We have used our implementation to conduct preliminary experiments. The results indicate that the abstraction is effective for the goal of the optimization based on quasi-equal clocks: it is coarse enough to yield a drastic reduction of the size of the zone graph. Still, the abstraction is precise enough to identify a large class of quasi-clocks.

Contributions

The key technical contributions that are described in this chapter are summarized as follows:

- We present the notion of *quasi-equal* clocks. Quasi-equal clocks are clocks which are always equal except at intervals of zero-time duration.
- We introduce the notion of *zero-time configuration*, *zero-time behavior* and a *relax operator* on zones. By using these notions we present an abstraction technique, *zero-time abstraction*. We show that that our abstraction technique is sound.
- We provide complexity results for our abstraction method. We show that the size of the concrete zone graph is an upper bound for the size of our abstraction.
- We present an algorithm for effectively computing quasi-equal clocks. We present a formal proof for the correctness of the algorithm.

Quasi-equal clocks are closely related to overclocks from the previous chapter. Reductions of quasi-equal clocks have been published in [76, 77]. The abstraction method

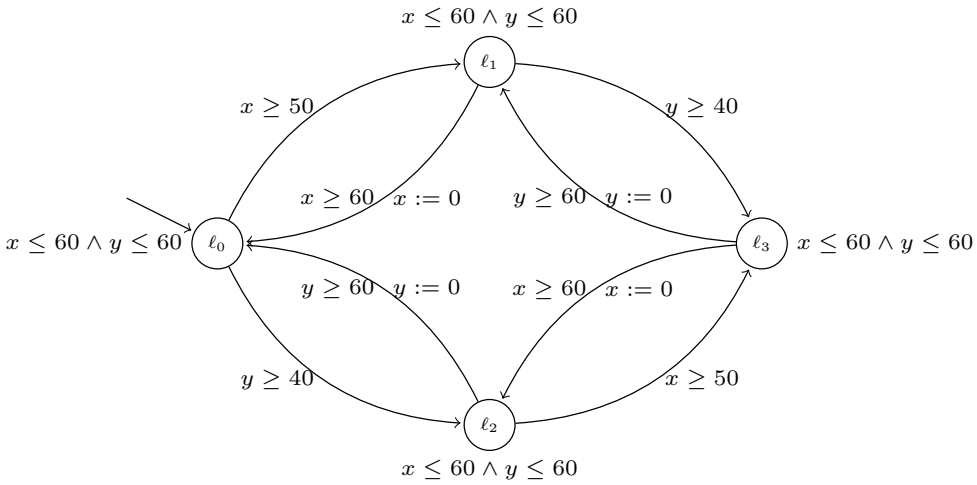


Figure 5.1: Timed automaton with quasi-equal clocks

that we present in this chapter has been published in [101]. In addition, we have successfully generalized the notion of quasi-clocks for timed automata to the notion of quasi-dependent variables for hybrid systems. We have successfully extended the abstraction technique from this chapter, to the context of hybrid systems. These results are published in [30].

Outline

In Section 5.1, we use an example to illustrate the notion of quasi-equal clocks and investigate the issue of zero time behavior. In Section 5.2, we formalize our abstraction by means of an abstract zone graph and a simulation relation. In Section 5.3, we give an upper bound for the size of the abstract zone graph. Then, we illustrate with an example the effectiveness of the abstraction, i.e. the reduction of size through the abstraction. In Section 5.4, we present an algorithm for computing the reachable abstract zone graph (on the fly). The output of the algorithm is a relation that identifies which clocks are quasi-equal.

5.1 Quasi-equal Clocks

Quasi-equal clocks are closely related to overclocks presented in Section 4.2. However, they are not semantically defined in terms of reset locations from Section 3.1 but in terms of time. Intuitively two clocks are quasi-equal if they are always equal except at some points of time. In general, this clock differences are introduced by the interleaving semantics of timed automata. Quasi-equal clocks can be simplified, (e.g. using the reduction methodsd presented in [76, 77]) while preserving most or all the system

properties. The result is often an exponential gain in the verification times.

In the case of TDMA based systems. Every component has its own clock. The clocks are initialized and synchronized. At the end of the TDMA cycle, clocks are restarted and continue to be synchronized and equal. Therefore, from an observer point of view these clocks appear to be equal. In reality, clocks will not be perfectly synchronous. However, studies (e.g. [88]) suggest that by having appropriate time guards, perfect clocks can be assumed. For the semantics of timed automata these clocks will be however not equal, but quasi-equal. This is because of the interleaving semantics. All clocks are reset at one point of time, but in an interleaving fashion, leading to different clock valuations in zero time. It is natural to observe that the system behavior can be preserved while using a reduced number of clocks.

For the rest of this chapter, let us fix a timed automaton $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$. In particular since for our abstraction method, actions are not relevant. We will disregard actions in edges. Further, we will work with symbolic configurations, consisting of pairs of locations and zones. We will use $Z \in \Phi(\mathbb{X})$ for some zone corresponding to $\mathcal{A}$.

Definition 5.1 (Quasi-equal Relation)**.** *The* quasi-equal *relation* $\equiv$ *for timed automaton* $\mathcal{A}$ *is the relation containing all pairs of clocks for which in all computations of* $\mathcal{A}$ *their values are equal, except at points of time where they are reset and time is not allowed to elapse. Formally, it is defined as:*

$$\equiv \stackrel{\mathsf{def}}{=} \{(x,y) \mid x,y \in \mathbb{X} \text{ and } \langle \ell_0, \{\nu_0\}\rangle \rightarrow^*_{ZG} \langle \ell, Z\rangle \implies \forall \nu \in Z.\ \nu \models x = y \vee x = 0 \vee y = 0\}$$

♢

Two clocks are in relation, if for all reachable configurations in the zone graph, their valuations are equal or one of the valuations is equal to zero. In [76] it has been shown that the quasi-equal relation is an equivalence relation.

Proposition 5.2. *The quasi-equal relation* $\equiv$ *is an equivalence relation.* ♢

The abstraction method that we present in this chapter, will enable us to efficiently compute the quasi-equal relation for a given automaton. Let us illustrate the notion of quasi-equal clocks with an example.

Example 5.3. (Quasi-equal clocks) Consider the timed automata presented in Figure 5.1 with clocks x and y. Clearly, clocks x and y are not equal since for example in the computation $\langle \ell_0, \{\nu_0\}\rangle \rightarrow^*_{ZG} \langle \ell_1, x = 60 \wedge y = 0\rangle$ the configuration $\langle \ell_1, x = 60 \wedge y = 0\rangle$ yields different values for clocks x and y. However, note that the invariant $I(\ell_1) = x \leq 60 \wedge y \leq 60$ will prevent time to elapse at this configuration and thus the only successor of this configuration is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60\rangle$

in which the values for clocks x and y are equal. Indeed, it is the case that for all computations from the automaton in Figure 5.1, the values for clocks x and y are either equal or the value of one clock is zero and a reset in zero time for the other clock occurs. Therefore, clocks x and y are quasi-equal. Clearly, the behavior of timed automata in Figure 5.1, can be simulated by using one clock, which yields an important speed up in the verification time. ♢

5.2 Zero Time Behavior Abstraction

In this section we present our method for detecting quasi-equal clocks. The main observation is that if two clocks x and y are quasi-equal then for all computations if one clock say x is reset then a reset for the other clock y must appear in some future configuration in the computation path. In particular, for all the configurations in the computation fragment between the resets of x and y time cannot elapse (i.e. all the delay successors of a configuration have a delay of zero). Another key observation is that if in a configuration time is allowed to elapse, clocks x and y must be strictly equal. Therefore, to detect quasi-equal clocks, we do not only need to consider resets of clocks but also configurations in which time is allowed to elapse and configurations in which time is not allowed to elapse.

As shown in Example 5.3, the zero time behavior of a timed automaton may cause quasi-equal clocks to arise. Therefore, the zero time behavior of an automaton is important for detecting quasi-equal clocks.

Example 5.4. (Zero Time Behavior) Consider Example 5.3. In the computation, $\langle \ell_0, \{\nu_0\}\rangle \rightarrow^*_{ZG} \langle \ell_1, x = 60 \wedge y = 0\rangle$, first a reset for clock y occurs, then in the next transition a reset for clock x occurs. In this case, the length of the zero time computation is just one transition. However, this does not need to be the case. Consider the computation $\langle \ell_0, \{\nu_0\}\rangle \rightarrow^*_{ZG} \langle \ell_0, x = 0 \wedge y \leq 60 \wedge y \geq 60\rangle$, the invariant $I(\ell_0) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_2, x = 0 \wedge y \leq 60 \wedge y \geq 60\rangle$. The invariant $I(\ell_2) = x \leq 60 \wedge y \leq 60$ will not let time elapse and the only successor is $\langle \ell_0, x = 0 \wedge y = 0 \wedge x \leq 60 \wedge y \leq 60\rangle$, where time may elapse but the values for clocks x and y are equal. In general, the number of transitions that may occur in zero time might be infinite. ♢

The following definition formalizes our notion of zero time configurations, i.e. configurations for which time cannot elapse.

Definition 5.5 (Zero Time Configuration)**.** *A configuration* $\langle \ell, Z\rangle \in SConf(\mathcal{A})$ *is* zero

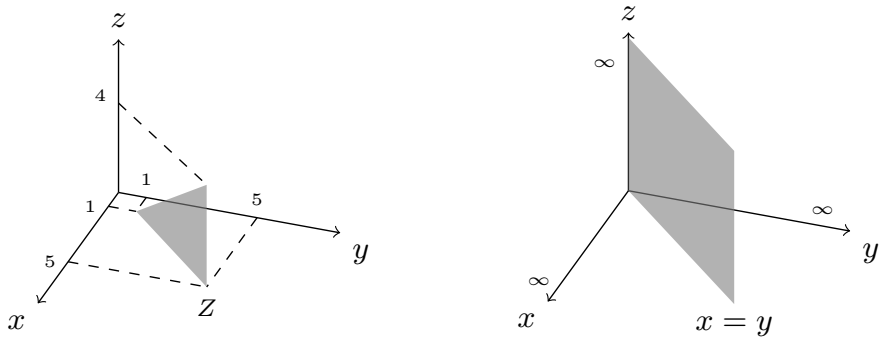


Figure 5.2: The relax operator on zone $Z := x-y \leq 0 \wedge x-y \geq 0 \wedge x \geq 1 \wedge y \leq 5 \wedge y-z \leq 1$.

time *if the invariant of location ℓ precludes time to elapse. Formally,*

$$\mathsf{zt}(\ell, Z) \stackrel{\mathsf{def}}{=} \forall \nu \in Z, d \in \mathbb{R}_0^+.\ \nu + d \models I(\ell) \implies d = 0.$$

$\diamondsuit$

Our method preserves as much as possible the information corresponding to zero time configurations and abstracts away much information from the non-zero time configurations.

If a configuration $\langle \ell, Z \rangle$ is zero time, our method will preserve all the information in Z. However, if the configuration $\langle \ell, Z \rangle$ is non-zero time, our method will abstract Z by means of the relax operator rlx to a much bigger zone $\mathsf{rlx}(Z)$, which preservers the strict equalities entailed by the zone Z. The following definition formalizes the relax operator.

Definition 5.6 (Relax Operator)**.** *Given a zone $Z \in \Phi(\mathbb{X})$. The* relax operator rlx *applied to the zone Z over-approximates Z by a conjunction of the clock equalities it entails. Formally,*

$$\mathsf{rlx}(Z) \stackrel{\mathsf{def}}{=} \bigwedge \{x = y \mid x, y \in \mathbb{X} \text{ and } \forall \nu \in Z.\ \nu \models x = y\}.$$

$\diamondsuit$

Example 5.7. (Relax Operator) Consider the zone Z from Figure 5.2 left. It is the case that all the valuations in Z satisfy the constraint $x = y$. However, there are valuations in Z which do not satisfy $x = z$ and also valuations which do not satisfy $y = z$. Therefore, the result of applying the relax operator to Z yields the zone $x = y$ as shown in Figure 5.2 right. Note, that the zone $\mathsf{rlx}(Z)$ has no constraints on the unequal

clocks. Another important property of $\mathsf{rlx}(Z)$ is that it contains only positive equalities and thus will remain unaffected by passage of time, that is $\mathsf{rlx}(Z) = \mathsf{rlx}(Z)^{\uparrow}$. Note, that $\mathsf{rlx}(Z)$ is much bigger than Z and it is closed with respect to the delay operator. $\diamondsuit$

If a configuration is zero time our method will abstract the configuration by means of the normalization $\mathsf{norm}_{k,\mathcal{G}}$ operator presented in Section 2.1.2. On the contrary, if a configuration is non-zero time, our method will abstract the configuration by means of the relax rlx operator. The normalization operator $\mathsf{norm}_{k,\mathcal{G}}$ is increasing, idempotent and yields a finite number of zones. Since our method relies on both the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$ and the relax operator rlx it is important that the relax operator exhibits the afore mentioned properties.

Lemma 5.8 (Properties Of The Relax Operator)**.** *The relax operator is increasing, idempotent and yields a finite abstraction. Formally, the following hold:*

1. $Z \subseteq \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
2. $\mathsf{rlx}(\mathsf{rlx}(Z)) = \mathsf{rlx}(Z)$ *for any* $Z \in \Phi(\mathbb{X})$
3. *the set* $\{\mathsf{rlx}(Z) \mid Z \in \Phi(\mathbb{X})\}$ *is finite.*

Proof. (1) By contradiction on the converse $\mathsf{rlx}(Z)^c \subset Z^c$. Assume $\nu \in \mathsf{rlx}(Z)^c$ and $\nu \notin Z^c$. which means that $\nu \in Z$. By definition of rlx, we have $\nu \in \mathsf{rlx}(Z)^c$ if ν does not satisfy a conjunct $x = y$ in $\mathsf{rlx}(Z)$ for some $x, y \in \mathbb{X}$. Since such a conjunct exists in $\mathsf{rlx}(Z)$ by definition of rlx we have that $\forall \nu' \in Z.\nu' \models x = y$ but $\nu \in Z$ and $\nu \models x \neq y$ which is a contradiction.

(2) By definition of the relax operator rlx, $\mathsf{rlx}(Z)$ is a conjunction of equalities such that $\forall \nu' \in Z.\nu' \models x = y$ for every equality in $\mathsf{rlx}(Z)$. Clearly, applying rlx to $\mathsf{rlx}(Z)$ will produce the same set of equalities since $\forall \nu' \in Z.\nu' \models x = y$ for every equality in $\mathsf{rlx}(Z)$.

(3) Let $|\mathbb{X}|$ denote the number of clocks in $\mathcal{A}$. Then the number of all possible relaxed zones, i.e. all possible equality combinations for the clocks in $\mathbb{X}$ is in $\mathcal{O}(2^{|\mathbb{X}|-1})$. $\square$

In Chapter 2 Section 2.1 we defined the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$. The normalization operator is used to produce a finite abstraction of the the system. Note that the finite zone graph obtained by using the normalization operator $\mathsf{norm}_{k,\mathcal{G}}$, is a sound and complete abstraction, i.e. abstracted zones preserve all the relevant information of the system. Our abstraction is a combination of both the normalization $\mathsf{norm}_{k,\mathcal{G}}$ operator and the relax operator rlx. We now define an abstraction function on symbolic configurations.

Definition 5.9 (Zero Time Abstraction Function)**.** *The abstraction function* α_{zt} *applied to a configuration* $\langle \ell, Z \rangle$ *preserves the information if the configuration is zero time and it abstracts it using the relax operator otherwise, i.e.*

$$\alpha_{\mathsf{zt}}(\langle \ell, Z \rangle) := \begin{cases} \langle \ell, \mathsf{norm}_{k,\mathcal{G}}(Z) \rangle & \text{if } \mathsf{zt}(\ell, Z) \\ \langle \ell, \mathsf{rlx}(Z) \rangle & \text{otherwise.} \end{cases}$$

5.2.1 Abstract Zone Graph

Our goal is to construct an abstract zone graph in which quasi-equalities can be soundly and efficiently computed. Given a timed automaton, our method will construct an abstract transition system, which preserves as much as possible the zero time behavior of a timed automaton. The configurations of the abstract transition system are pairs consisting of locations and zones. The zones that we compute are of two types. Either a zone for which time is guaranteed not to elapse or a conjunction of equalities for clocks for which time may elapse. We now continue with the formal definition of our method.

Definition 5.10 (Abstract Zone Graph)**.** *Timed automaton* $\mathcal{A}$ *induces the* abstract zone graph

$$ZG^{\#}(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow^{\#}_{ZG}, c_0^{\#})$$

where:

- $SConf(\mathcal{A}) \subseteq L \times \Phi(\mathbb{X})$ *is the set of configurations*
- $c_0^{\#} = \alpha_{\mathsf{zt}}(\langle \ell_0, \{\nu_0\} \rangle)$ *is the initial configuration*
- $\rightarrow^{\#}_{ZG} \subseteq SConf(\mathcal{A}) \times SConf(\mathcal{A})$ *is the transition relation. Given a configuration* $\langle \ell, P \rangle \in SConf(\mathcal{A})$*, there is a transition* $\langle \ell, P \rangle \rightarrow^{\#}_{ZG} \alpha_{\mathsf{zt}}(\langle \ell', F \rangle)$ *if there is an edge* $(\ell, \varphi, Y, \ell') \in E$ *and* F *is not empty. Where* $F = (P \wedge \varphi \wedge I(\ell))[Y := 0] \wedge I(\ell')$

$\diamondsuit$

The initial abstract configuration is a conjunction asserting all clocks to be equal. Given a configuration and an edge, the successor zone F is computed. By definition of the abstraction function α_{zt}, if the destination configuration is zero time, the zone F will be normalized to $\mathsf{norm}_{k,\mathcal{G}}(F)$. This is because in zero time it is important to preserve as much information as possible. This information is useful for detecting zero time paths in which quasi-equal clocks may arise. If the destination configuration is not zero time, the zone F will be relaxed to $\mathsf{rlx}(F)$. The zone $\mathsf{rlx}(F)$ consist of conjuncts asserting all strict equalities of clocks in zone F. In addition, the relax operator will remove inequalities introduced by clock resets. Note that the abstract zone graph only performs discrete

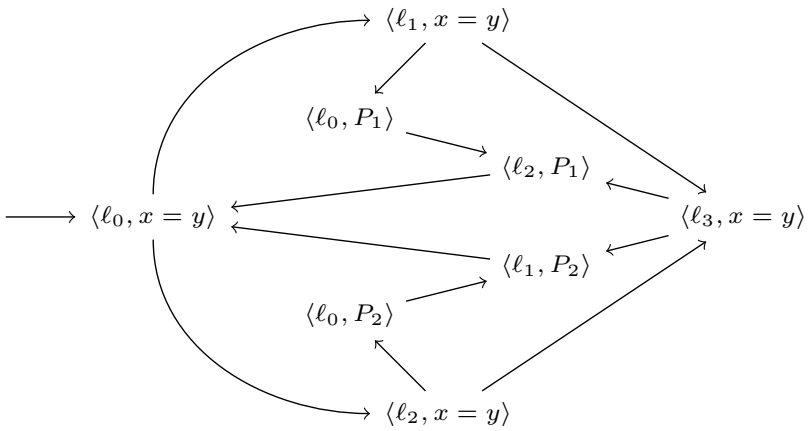


Figure 5.3: Abstract zone graph corresponding to TA in Figure 5.1.

transitions. This is because if a configuration is "detected" as non-zero time, the relax operator will be applied and the corresponding zone is closed with respect to the delay operator.

We use $\mathsf{norm}_{k,\mathcal{G}}(F)$ to ensure the relation $\rightarrow^{\#}_{ZG}$ to be finite. We remind the reader that any normalization operator norm' such that $Z \subseteq \mathsf{norm}'(Z)$, $\mathsf{norm}'(\mathsf{norm}'(Z)) = \mathsf{norm}'(Z)$ and the set $\{\mathsf{norm}'(Z) \mid Z \in \Phi(\mathbb{X})\}$ is finite, will be suitable for our abstraction method.

Example 5.11. Figure 5.3 shows the abstract zone graph corresponding to the automaton in Figure 5.1, where zones P_1, P_2 are $P_1 := x \leq 60 \wedge x \geq 60 \wedge y = 0$ and $P_2 := y \leq 60 \wedge y \geq 60 \wedge x = 0$. Note, that in configurations where time may elapse the corresponding zone is $x = y$. In the transition induced by the edge $(\ell_1, x \geq 60, x := 0, \ell_0)$ from the automaton in Figure 5.1 and configuration $\langle \ell_1, P_1 \rangle$, the successor configuration $\langle \ell_0, P' \rangle$ with zone $P' = (P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is not zero time. Therefore, $(P_1 \wedge x \geq 60)[\{x\} := 0] \wedge I(\ell_0)$ is relaxed to $x = y$, leading to configuration $\langle \ell_0, x = y \rangle$. Since, every zone in the set of reachable configurations implies that $x = y \vee x = 0 \vee y = 0$, our abstraction allow us to soundly conclude that x and y are quasi-equal.

$\diamondsuit$

For a timed automaton $\mathcal{A}$, we formalize the behavior of the corresponding abstract zone graph $ZG^{\#}(\mathcal{A})$ with respect to the zone graph $ZG(\mathcal{A})$ via a simulation relation.

Definition 5.12 (Simulation Relation)**.** *Given a timed automaton $\mathcal{A}$, a simulation relation for the zone graph $ZG(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow_{ZG}, c_0)$ and the abstract zone graph $ZG^{\#}(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow^{\#}_{ZG}, c_0^{\#})$, is a binary relation $\preccurlyeq$ on $SConf(\mathcal{A})$ such that:*

1. $c_0 \preccurlyeq c_0^{\#}$
2. *if* $\langle \ell, Z \rangle \preccurlyeq \langle \ell_1, P \rangle$ *then:*
   a) $\ell = \ell_1$ *and* $Z \subseteq P$
   b) *if* $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell', Z' \rangle$ *with edge* $(\ell, \varphi, Y, \ell') \in E$, *then there exists* $\langle \ell', P' \rangle$ *such that* $\langle \ell, P \rangle \rightarrow_{ZG}^{\#} \langle \ell', P' \rangle$ *with edge* $(\ell, \varphi, Y, \ell')$ *and* $\langle \ell', Z' \rangle \preccurlyeq \langle \ell', P' \rangle$
   c) *if* $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$, *then* $\langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle \preccurlyeq \langle \ell, P \rangle$.

If a simulation relation $\preccurlyeq$ *exists, we say that the abstract zone graph* $ZG^{\#}(\mathcal{A})$ *simulates the zone graph* $ZG(\mathcal{A})$. ♢

In the rest of the paper we will consistently use Z and P to refer to zones in the zone and abstract zone graph respectively. The definition of simulation relation given above is quite specific. This allow us to better explain the relation between the zone graph and the abstract zone graph. If two configurations are in relation, the zone in the abstract zone graph is always bigger or equal than the corresponding zone in the zone graph. If there is a discrete transition in the zone graph then there is a discrete transition in the abstract zone graph as well. If there is a delay transition in the zone graph, meaning that the configuration is non-zero time, the abstract zone graph "remains" at its current configuration. In the latter case, since zones in the abstract zone graph are bigger or equal than zones in the zone graph, the corresponding abstract configuration is also non-zero time and closed with respect to delays. The following lemma guarantees the existence of a simulation relation.

Lemma 5.13. *For a timed automaton* $\mathcal{A}$, *the abstract zone graph* $ZG^{\#}(\mathcal{A})$ *simulates the zone graph* $ZG(\mathcal{A})$.

Proof. We will show the existence of a simulation relation $\preccurlyeq$ by inductively constructing one. First let $\preccurlyeq_0 = \{c_0, \alpha_{\mathsf{zt}}(c_0)\}$. Therefore, initial configurations are in relation and condition 1. is satisfied. For the inductive step, let $(\langle \ell, Z \rangle, \langle \ell, P \rangle) \in \preccurlyeq_n$. By I.H. we can assume that $Z \subseteq P$. We now need to consider the transitions in the concrete zone graph:

- Discrete transitions. There is a transition $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell', Z' \rangle$ with edge $(\ell, \varphi, Y, \ell') \in E$. Then by definition of the abstract transition relation we have, $\langle \ell, P \rangle \rightarrow_{ZG}^{\#} \langle \ell', P' \rangle$ with edge e. By definition of $\rightarrow_{ZG}$ we have $Z' = (Z \wedge \varphi)[Y := 0] \wedge I(\ell')$ and by definition of $\rightarrow_{ZG}^{\#}$ we have $F = (P \wedge \varphi)[Y := 0] \wedge I(\ell')$. Note, that since $Z \subseteq P$ it is the case that $(Z \wedge \varphi)[Y := 0] \subseteq (P \wedge \varphi)[Y := 0]$. An thus $Z' \subseteq F$. Now, according to $\rightarrow_{ZG}^{\#}$ there are two possible values for P' (i.e. F and $\mathsf{rlx}(F)$):
  - $\mathsf{zt}(\ell', F) = \top$ then $P' = \mathsf{norm}_{k,\mathcal{G}}(F)$ and $P' \supseteq Z'$ since $Z \subseteq F \subseteq \mathsf{norm}_{k,\mathcal{G}}(F)$.

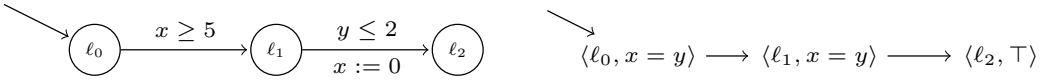


Figure 5.4: Abstraction is sound but not complete. Left: an automaton with two equal clocks x and y. Right: the abstract zone graph corresponding to the automaton at the left. Note, that in the abstract zone graph clocks x and y at configurations in location ℓ_2 are not equal. This is because at location ℓ_1 the abstraction is too coarse yielding the predicate $x = y$. Thus, the guard $y \leq 2$ is satisfied and the reset to x is executed, causing clocks x and y to be unequal.

  - $\mathsf{zt}(\ell', F) = \bot$ then $P' = \mathsf{rlx}(F)$. By Lemma 5.8, we know that the rlx operator is monotone. Thus, we have that $Z' \subseteq F \subseteq P' = \mathsf{rlx}(F)$.

  For all the above cases we have that $(\langle \ell', P' \rangle), \langle \ell', Z' \rangle)$ satisfies conditions under (2). Let $\preccurlyeq_{n+1} = \preccurlyeq_n \cup \{(\langle \ell', P' \rangle), \langle \ell', Z' \rangle)\}$

- Delay transitions. There is a transition $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell, Z^{\uparrow} \wedge I(\ell) \rangle$. Since there is a delay transition, the configuration is no zero time, i.e. $\mathsf{zt}(\langle \ell, Z \rangle) = \bot$ since $Z \subseteq P$ it follows that $\mathsf{zt}(\langle \ell, P \rangle) = \bot$ and P is a relaxed zone closed under time passage. Let $\preccurlyeq_{n+1} = \preccurlyeq_n \cup \{(\langle \ell^{\uparrow} \wedge I(\ell), Z \rangle, \langle \ell, P \rangle)\}$

Let $\preccurlyeq$ be the union of all $\preccurlyeq_i$ for $i \in \mathbb{N}$. Since all elements in $\preccurlyeq$ satisfy the conditions under (2) and $\preccurlyeq$ satisfies condition (1). We conclude that $\preccurlyeq$ is a simulation relation and thus $ZG(\mathcal{A}) \preccurlyeq ZG^{\#}(\mathcal{A})$. □

Our goal is to find the set of quasi-equal clocks for a given timed automaton $\mathcal{A}$. We now define the quasi-equal relation induced by the abstract zone graph $ZG^{\#}(\mathcal{A})$ as the set of quasi-equalities implied by all the zones in the set of its reachable configurations.

Definition 5.14 (Abstract Quasi-equal Relation)**.** *Given timed automaton $\mathcal{A}$. The abstract quasi-equal relation $\equiv^{\#}$ induced by the abstract zone graph $ZG^{\#}(\mathcal{A})$, is the set of pairs of clocks such that for all pairs, their values in the reachable configurations are equal or one clock is equal to zero. Formally,*

$$\equiv^{\#} \stackrel{\mathsf{def}}{=} \{(x, y) \mid x, y \in \mathbb{X} \text{ and } c_0^{\#}(\rightarrow_{ZG}^{\#})^{*}\langle \ell, P \rangle \implies P \subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}\}.$$

♢

Our method is sound in the sense that if two clocks are quasi-equal in the abstract zone graph, then they are quasi-equal in the concrete zone graph. Our method is not complete in the sense that if two clocks are quasi-equal in the zone graph, then they might not be quasi-equal in the abstract zone graph. We illustrate the non-completeness of our method with an example.

Example 5.15. As an example consider the automaton in Figure 5.4. In the automaton we have that $x \equiv y$. Further, x is strictly equal to y. Note that in the corresponding abstract zone graph in Figure 5.4 right. Clocks x and y are not equal at location ℓ_2, i.e. $x \not\equiv^{\#} y$. This is because the edge $e = (\ell_1, x \leq 2, \{x\}, \ell_2)$ is an unfeasible edge, i.e. it does not induce a reachable transition in the zone graph. Edge e will cause clocks x, y to be unequal in the abstract zone graph. Note, that at location ℓ_1 the clocks x and y are equal, and their valuations are greater or equal than 5. Therefore, the guard $y \leq 2$ is never satisfied, and the clock x is never reset. Thus, clocks remain equal. Since configurations at location ℓ_1 are not zero time. The abstract zone graph will summarize all of this configurations by the predicate $x = y$. At computing the abstract successor given the zone $x = y$, the guard $y \leq 2$ is satisfied and the reset of x is executed. Since the configuration at location ℓ_2 is not zero time and there is no reset for clock y, the relax operator will make clocks x and y unequal. ♢

Surprisingly, we have not been able to find an example for which the abstraction is not complete given that the considered timed automaton contains only feasible edges. The following theorem states formally the soundness of the abstraction.

Theorem 5.16 (Zero Time Abstraction is Sound)**.** *Given timed automaton $\mathcal{A}$. If two clocks are quasi-equal in the abstract zone graph $ZG^{\#}(\mathcal{A})$, then they are quasi-equal in the zone graph $ZG(\mathcal{A})$. Formally,*

$$\forall x, y \in \mathbb{X}.\ x \equiv^{\#} y \implies x \equiv y.$$

Proof. Given $x \equiv^{\#} y$, by definition of the abstract quasi-equal relation $\equiv^{\#}$ the following holds:

$$\forall \langle \ell, P \rangle \in \mathit{SConf}(\mathcal{A}).\ q_0(\rightarrow^{\#}_{ZG})^* \langle \ell, P \rangle \implies P \subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}.$$

We continue by contradiction. Assume $x \not\equiv y$, then by definition of the quasi-equal relation $\equiv$ we have that

$$\exists \langle \ell, Z \rangle \in \mathit{SConf}(\mathcal{A}).\ c_0 \rightarrow_{ZG} \langle \ell, Z \rangle \wedge Z \not\subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}.$$

Which means that there exist a valuation $\nu \in Z$, such that $\nu \not\models x = y \vee x = 0 \vee y = 0$. By Lemma 5.13, we have that $ZG(\mathcal{A}) \preccurlyeq ZG^{\#}(\mathcal{A})$. Therefore, there exist a $\langle \ell, P \rangle \in SConf(\mathcal{A})$ s.t. $(\langle \ell, Z \rangle, \langle \ell, P \rangle) \in \preccurlyeq$. By definition of the simulation relation $\preccurlyeq$ it holds that $Z \subseteq P$. Finally, we have that $\nu \in Z$ and $\nu \notin P$ which is a contradiction. □

For implementation purposes, it is important that the relation $\rightarrow^{\#}_{ZG}$ is finite. Given a timed automaton $\mathcal{A}$ and by definition of the transition relation $\rightarrow^{\#}_{ZG}$. A configuration has two possible successors. Either a successor corresponding to the application of the $\mathsf{norm}_{k,\mathcal{G}}$ operator, or a successor corresponding to the application of the rlx operator. The set of zones generated by $\mathsf{norm}_{k,\mathcal{G}}$ is finite. Further, the set of zones generated by rlx is in $\mathcal{O}(2^{|\mathbb{X}|})$, since it contains only positive equalities for the clocks in $\mathbb{X}$. Thus, we obtain the following theorem.

Theorem 5.17. *The transition relation $\rightarrow^{\#}_{ZG}$ is finite.*

Proof. By definition of $\rightarrow^{\#}_{ZG}$ there are two possible successors for a zone P. Namely, $\mathsf{norm}_{k,\mathcal{G}}(F)$ and $\mathsf{rlx}(F)$. By definition of $\mathsf{norm}_{k,\mathcal{G}}$ we have that the set $\{\mathsf{norm}_{k,\mathcal{G}}(F) \mid F \in \Phi(\mathbb{X})\}$ is finite. By Lemma 5.8 the zones generated by the rlx operator are finite. Therefore, the reachable zones in $ZG^{\#}(A)$ are finite which implies that $\rightarrow^{\#}_{ZG}$ is also finite. □

5.3 Complexity

In this section we characterize the size of the abstract zone graph based on the number of reachable configurations. We show that when an automaton $\mathcal{A}$ contains only feasible edges i.e. for each edge e in the timed automaton there exists a computation path in which the edge e induces a transition, then the size of the zone graph $ZG(\mathcal{A})$ is an upper bound for the size of the abstract zone graph $ZG^{\#}(\mathcal{A})$.

Definition 5.18 (Size)**.** *For timed automaton $\mathcal{A}$ we defined the size of the zone graph $ZG(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow_{ZG}, c_0)$ and the size of the abstract zone graph $ZG^{\#}(\mathcal{A}) = (Conf(\mathcal{A}), \rightarrow^{\#}_{ZG}, c^{\#}_0)$ to be the number of reachable configurations. Formally,*

$$|ZG(\mathcal{A})| = |\{c \mid c_0 \rightarrow^{*}_{ZG} c\}|$$

and

$$|ZG^{\#}(\mathcal{A})| = |\{c \mid c^{\#}_0 (\rightarrow^{\#}_{ZG})^{*} c\}|$$

respectively. ♢

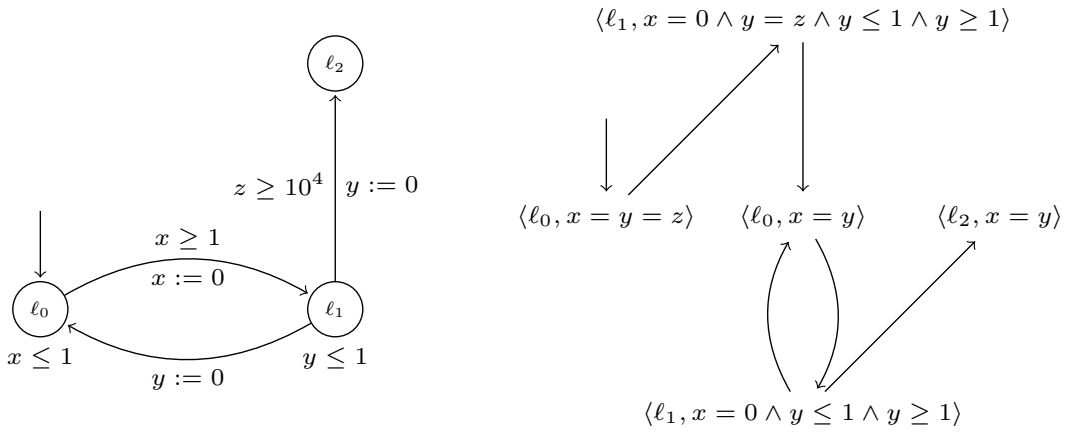


Figure 5.5: Left: a timed automaton with clocks, x, y and z. Clocks x and y are quasi-equal. Right: the corresponding abstract zone graph.

The following theorem shows that when in the input automaton all edges induce a reachable configuration in the zone graph $ZG(\mathcal{A})$. Then the number of configurations from the corresponding abstract zone graph $ZG^{\#}(\mathcal{A})$ is smaller or equal than the number of configurations in the zone graph $ZG(\mathcal{A})$. The theorem follows from the following facts. First, if the abstract zone graph $ZG^{\#}(\mathcal{A})$ performs an action transition with edge e then the zone graph $ZG(\mathcal{A})$ also performs a transition with edge e. Second, by Lemma 5.13 it is the case that $ZG^{\#}(\mathcal{A})$ simulates $ZG(\mathcal{A})$ and all zones in the configurations of $ZG^{\#}(\mathcal{A})$ are bigger than the corresponding ones in $ZG(\mathcal{A})$.

Theorem 5.19 ($|ZG^{\#}(\mathcal{A})| \leq |ZG(\mathcal{A})|$)**.** *Given a timed automaton $\mathcal{A}$, the zone graph $ZG(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow_{ZG}, c_0)$ and the abstract zone graph $ZG^{\#}(\mathcal{A}) = (SConf(\mathcal{A}), \rightarrow^{\#}_{ZG}, c_0^{\#})$. If $\mathcal{A}$ is such that for all $e = (\ell, \varphi, Y, \ell') \in E$ there is a transition $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell', Z' \rangle$ with edge e and $c_0 \rightarrow^*_{ZG} \langle \ell, Z \rangle$ then $|ZG^{\#}(\mathcal{A})| \leq |ZG(\mathcal{A})|$.*

Proof. We show that whenever there is a transition in $ZG^{\#}(\mathcal{A})$ then there is also a transition in $ZG(\mathcal{A})$ and that all the zones in $ZG^{\#}(\mathcal{A})$ are bigger than their correspondent ones in $ZG(\mathcal{A})$. Thus we can conclude $|ZG^{\#}(\mathcal{A})| \leq |ZG(\mathcal{A})|$. Formally, given a transition $\langle \ell, P \rangle \rightarrow^{\#}_{ZG} \langle \ell', P' \rangle$, by definition of $\rightarrow^{\#}_{ZG}$ there exist an edge $e \in E$ and since by assumption for all edges $e = (\ell, \varphi, Y, \ell') \in E$ there is a transition $\langle \ell, Z \rangle \rightarrow_{ZG} \langle \ell', Z' \rangle$ with edge e such that $c_0 \rightarrow^*_{ZG} \langle \ell, Z \rangle$. We have that $ZG(\mathcal{A})$ has also performed a transition. In addition, by the Simulation Lemma 5.13, we have that $ZG^{\#}(\mathcal{A}) \preccurlyeq ZG(\mathcal{A})$. Thus, we have that for all $\langle \ell, Z \rangle \preccurlyeq \langle \ell, P \rangle$, $Z \subseteq P$. Therefore, $|ZG^{\#}(\mathcal{A})| \leq |ZG(\mathcal{A})|$. □

The feasibility of edges is not a strong condition, since in practice unfeasible edges will not occur intentionally, except in the cases when the modeler makes a mistake.

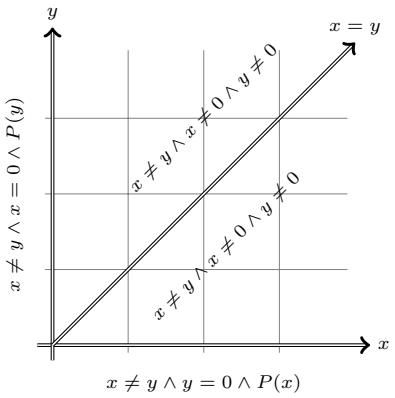


Figure 5.6: Quasi-equal zones for clocks x, y. Where $P(x)$, $P(y)$ is a clock constraint over x and y respectively.

Theorem 5.19 gives an upper bound on the size of the abstract zone graph. In practice for a timed automaton the difference on the size of the abstract and the size of the concrete system can be exponential. We illustrate this fact in the following example.

Example 5.20. Consider the automaton $\mathcal{A}$ in Figure 5.5 left. We observe that the zero time behavior occurs at points of time where $x = 1 \wedge y = 1 \wedge z = n$ with $n \in \{1, 2, \ldots, 10^6\}$. The normalized zone graph using maximal constant over approximation will contain at least 10^6 configurations. In Figure 5.5 right the complete full abstract zone graph $ZG^{\#}(\mathcal{A})$ is illustrated. At point of time $x = 1 \wedge y = 1 \wedge z = 1$ our method detects a zero time configuration and computes the exact successor zone $Z_1 := x = 0 \wedge y = z \wedge y \leq 1 \wedge y \geq 1$. Note that $\mathsf{zt}(\ell_1, Z_1)$ is valid, then there is a transition with edge $(\ell_1, \top, \{y\}, \ell_0)$. Our method computes $F := x = 0 \wedge y = 0 \wedge z = 1$ and the formula $\mathsf{zt}(\ell_0, F)$ is not valid, thus F is relax to $\mathsf{rlx}(F) := x = y$ leading to configuration $\langle \ell_0, x = y \rangle$. Since F does not imply a quasi-equality for clock z, the clock z is abstracted away. The resulting abstraction has only 5 configurations. ♢

5.3.1 Quasi-equal Zones

For theoretical reasons it is also interesting to see, how many zones are needed in order to detect quasi-equal clocks. The quasi-equal relation $\equiv$ is an equivalence relation and thus induces a partition. We call the corresponding equivalence classes quasi-equal zones (two clocks are quasi-equal or not in a zone). The set of quasi-equal zones is a subset of the zones of the corresponding zone graph. In the worst case, a zone is a region. Regions are used used to construct the region automaton as shown in [105]. The following lemma gives an upper bound on the number of quasi-equal zones.

Lemma 5.21. *Given a timed automaton $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$, let $k : \mathbb{X} \to \mathbb{Q}_0^+$ be a function mapping each clock x to the maximal constant $k(x)$ appearing in the guards or invariants in $\mathcal{A}$ containing x and $c = \max\{k(x) \mid x \in \mathbb{X}\}$. Then, the number of quasi-equal zones induced by the equivalence relation $\equiv$ is at most*

$$|\mathbb{X}| \cdot (2c + 2) + 2^{(|\mathbb{X}|-1)} - |\mathbb{X}|.$$

Proof. The proof is similar to the one presented in [105], page 160. The first summand $|\mathbb{X}| \cdot (2c+2)$ corresponds to the intervals $[x = 0], [0 < x < 1], [x = 1], [1 < x < 2], \ldots, [x = c], [x > c]$. Since for clocks to be quasi-equal they can be either equal or be 0. We only need to consider the intervals which are the axis of the euclidean space for $|\mathbb{X}|$. Therefore, we obtain the term $|\mathbb{X}| \cdot (2c + 2)$. The second and third term $2^{(|\mathbb{X}|-1)} - |\mathbb{X}|$ correspond to all regions, that are lines which represent some clocks to be equal and some clocks to be different than zero, for example for clocks x, y, z we could have $x = y \wedge z \neq 0$. The term $2^{(|\mathbb{X}|-1)}$ corresponds to all possible combinations of clocks being equal or being equal to zero, we substract $|\mathbb{X}| - 1$ because we have already counted the combinations for which one clock is not zero and the others are zero i.e. being in an axis. Finally, we substract the number of axis $|\mathbb{X}|$ given us the above formula, because they were already counted in the first term. □

In the worst case, the abstraction will enumerate all the possible quasi-equal zones. Clearly, the number of zones that we need to consider for finding quasi-equalities is smaller than the full number of zones from the corresponding zone automaton. Namely, $\mathcal{O}((2c + 2)^{|\mathbb{X}|} \cdot (4c + 3)^{\frac{1}{2} \cdot |\mathbb{X}| \cdot (|\mathbb{X}|-1)})$ as given in [105].

5.4 Algorithm

In this section, first we present a high level algorithm for performing a reachability analysis on the abstract zone graph induced by a given timed automaton. We have implemented the algorithm in our tool sAsEt and subsequently performed several experiments. We will discuss details of our implementation and experiments in Chapter 6.

Algorithm 5.1 lists a high level algorithm for finding quasi-equal clocks in a given timed automaton. For a timed automaton $\mathcal{A}$, the idea of Algorithm 5.1 is to traverse the reachable state space of $ZG^{\#}(\mathcal{A})$ while maintaining a relation QE containing quasi-equal clocks. The reachable state space is computed on the fly.

We continue by describing Algorithm 5.1 in detail. At line 2 the set QE is initialized to have all non reflexive quasi-equalities. At line 3 the while condition ensures that the algorithm will terminate either when the whole reachable space of $ZG^{\#}(\mathcal{A})$ has been visited or when there are no quasi-equal clocks in the relation QE. At lines 4 to 8

Algorithm 5.1 High level algorithm for detecting quasi-equal clocks

Input: timed automaton $\mathcal{A} = (L, \Sigma, \mathbb{X}, I, E, \ell_0)$

Output: a binary relation $QE \subseteq \mathbb{X} \times \mathbb{X}$ containing quasi-equal clocks in $\mathcal{A}$

1: $W := \{c_0^{\#}\}$, $V := \emptyset$
2: $QE := \{(x, y) \mid x, y \in \mathbb{X}$ and x is different than $y\}$
3: **while** $W \neq \emptyset$ and $QE \neq \emptyset$ **do**
4: take $\langle \ell, P \rangle$ from W
5: **for all** $(x, y) \in QE$ **do**
6: **if** $P \not\subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}$ **then**
7: $QE := QE \setminus \{(x, y)\}$
8: **end if**
9: **end for**
10: **if** $P \not\subseteq P'$ for all $\langle \ell, P' \rangle \in V$ **then**
11: add $\langle \ell, P \rangle$ to V
12: **for all** $\langle \ell', P' \rangle$ with $\langle \ell, P \rangle \rightarrow_{ZG}^{\#} \langle \ell', P' \rangle$ **do**
13: add $\langle \ell', P' \rangle$ to W
14: **end for**
15: **end if**
16: **end while**
17: **return** QE

the algorithm picks a configuration $\langle \ell, P \rangle$ to be explored and checks that all the quasi-equalities (x, y) in QE are implied by P. If this is not the case, then it will remove (x, y) from QE. Note that in the algorithm the size of QE only decreases. Finally, at lines 10 to 14 the algorithm computes the successor of $\langle \ell, P \rangle$ using the transition relation $\rightarrow_{ZG}^{\#}$.

Note, that all the operations needed in Algorithm 5.1 can be implemented using difference bound matrices [21, 61, 141] and thus our approach can be implemented in tools like Kronos [140] or Uppaal [91]. The check in line 6 for two clocks can be implemented by checking independently whether P satisfies any disjunct in $x = y \vee x = 0 \vee y = 0$. For computing the successor in line 12 by definition of $\rightarrow_{ZG}^{\#}$ it is necessary to compute $\mathsf{zt}(\ell', F)$ where F is a convex zone. This can be computed by checking the inclusion $F \wedge I(\ell') \supseteq F^{\uparrow} \wedge I(\ell')$.

While the use of linear real arithmetic and SMT solvers in our tool is expensive for difference logic constraints. It is also a flexible approach. By slightly modifying our implementation, we have been able to use SMT solvers for solving more expressive constraints which arise from the verification of hybrid systems [3].

If the set QE is never empty, Algorithm 5.1 computes the reachable space of $ZG^{\#}(\mathcal{A})$

and if a pair of clocks (x, y) is in QE by Theorem 5.16 it follows that they are quasi-equal in $\mathcal{A}$. The following theorem states that all clocks in the return value QE of Algorithm 5.1, are quasi-equal clocks.

Theorem 5.22 (Partial Correctness)**.** *For any timed automaton $\mathcal{A}$, if two clocks are in relation in the return value QE from Algorithm 5.1, then they are quasi-equal in the corresponding zone graph. Formally,*

$$\forall x, y \in \mathbb{X}.(x, y) \in QE \implies x \equiv y.$$

Proof. As long as $QE \neq \emptyset$, Algorithm 5.1 will generate the reachable configurations in $ZG^{\#}(\mathcal{A})$. Since $(x, y) \in QE$, the following holds: $\forall \langle \ell, P \rangle \in \mathit{Conf}(\mathcal{A})$. $(c_0^{\#}(\rightarrow_{ZG}^{\#})^*\langle \ell, P \rangle) \implies P \subseteq \{\nu \mid \nu \models x = y \vee x = 0 \vee y = 0\}$. That is $x \equiv^{\#} y$, by Theorem 5.16 it follows that $x \equiv y$. □

Algorithm 5.1 will terminate whenever the relation QE is empty or whenever the wait list W is empty. By Theorem 5.17, the relation $\rightarrow_{ZG}^{\#}$ is finite, meaning that the list W will be eventually empty. Thus we obtain the following theorem.

Theorem 5.23 (Termination)**.** *Algorithm 5.1 terminates for all inputs.*

Proof. The algorithm terminates if the set QE is empty. However, if there are quasi-equal clocks, the set QE will not be empty. In this case, he algorithm terminates, because the relation $\rightarrow_{ZG}^{\#}$ is finite as stated in Theorem 5.17. □

5.5 Related Work

In [54] an abstraction method is proposed for detecting *equal* clocks (at a particular location). Once equal clocks at a particular location have been detected, a substitution method can be used to reduce the number of clocks in that location. Thus, the method can effectively reduce the number of clocks per location and also in the whole timed automaton. In this sense, the method [54] is similar wrt. its goal to our method. The difference lies in technical details. Computing quasi-equal clocks requires to detect zero time paths. For this, our method tracks the actual valuations of clocks, which the method [54] does not. As a consequence, it will not be able to detect quasi-equal clocks when they are not equal.

Our approach is based on the zone abstraction method. Therefore, we produce a finite system consisting of pairs of locations and zones. Zones are predicates over clocks. Predicate abstraction [69, 28, 120] is used to compute finite approximations of infinite transition systems. In [125] predicate abstraction is used in the context of timed automata. This approach is an interesting alternative for computing our abstraction. The set

of predicates could be a subset of the predicates corresponding to the quasi-equal zones given in Section 5.3. Then, an abstraction refinement process will be needed. Note, that our definition of the abstract zone graph is constructive, i.e. the abstraction can be directly computed from the given automaton and the initial configuration.

In the previous chapter published in [100], a syntactical pattern for timed automata is proposed. Automata constructed under this pattern are called sequential timed automata. There, the so-called master clocks will be reset at exactly the same point of time. Thus, master clocks are quasi-equal by construction. Unfortunately, the class of sequential timed automata is rather restricted. Therefore, one would like to be able to use the general class of timed automata and apply a method for detecting or checking quasi-equal clocks. This motivates the use of quasi-equal clocks.

In [30] the notion quasi-dependent variables for hybrid systems is presented. For two quasi-dependent variables, their valuations are always related via a function except at some intervals of time of zero duration. Quasi-equal clocks are related by the identity function. Therefore, quasi-dependent variables are a generalization of quasi-equal clocks. We have successfully extended the abstraction method presented in this chapter for detecting quasi-dependent variables in hybrid systems.

The classical maximal constant abstraction used for construction the finite zone graph [27]. May yield a large number of zones, if the maximal constant is a large number. Many of these zones are not relevant for some local behavior, since the maximal constant is not relevant at all locations. In [17, 16] different abstractions for timed automata are presented. These abstractions have as goal to produce coarser, sound and complete abstractions by choosing relevant constants from the guards. These methods are complementary to our method, since we can always combine our analysis with an appropriate normalization operator.

Chapter 6

Proof of Concept

Contents

In this chapter, we show the applicability of the concepts presented in this thesis. First, in order to produce experimental results we present our tool sAsEt for analyzing timed systems. sAsEt is implemented in top of the Jahob [142, 114, 136] verification system. We have used our tool to compute the benchmarks of detecting quasi-equal clocks in timed automata [101] and quasi-dependent variables in hybrid systems [30]. Later in this chapter, we present our techniques applied to the fired alarm system from Section 1.3. By reducing quasi-equal clocks the verification times for the fire alarm system go from exponential to quadratic in the number of components(sensors) [76]. In addition, for the fire alarm system we show that the use of the sequential composition operator from Chapter 4, further improves the verification times to linear in the number of components [100].

Outline

In Section 6.1, we present our tool sAsEt for analyzing hybrid systems. We give an overview of the structure of the system. We then discuss important implementation details. Next, we present an example on how to use our tool for detecting quasi-equal clocks. In Section 6.2, we use our tool for detecting quasi-equal clocks in the simplified

fire alarm system presented in Section 1.3. We run a number of experiments in both sAsEt and Uppaal and show the corresponding results. In Section 6.3, we apply the techniques presented in this thesis to the industrial version of the simplified fire alarm system. Our results show that the verification times improve from exponential to linear in the number of components.

6.1 A Tool for Analyzing Timed Automata

As a proof of concept we have implemented our approach in our prototype tool sAsEt. sAsEt is implemented in top of the Jahob verification system. Our tool uses many of the libraries implemented in Jahob, e.g. the form library. Among other functions, the form library allows us to represent predicates as higher order logic (HOL [130]) formulae, rewrite higher order logic formulae to the SMT-LIB [13] standard, or use type inference [78].

Our implementation represents zones as linear real arithmetic formulae. As sAsEt constructs the abstract zone graph a number of constraints will arise. sAsEt will use an SMT solver to discharge them. In our experiments we used the solver Z3 [56]. The use of other solvers, e.g. CVC3 [14] is easy to implement, since this is already provided by the Jahob verification system.

In the following. First, we give an overview of the structure of the tool. Then, we will discuss some important implementation details. Finally, we will present an example on how to use sAsEt for computing quasi-equal clocks.

6.1.1 Design

Figure 6.1 illustrates the structure of the sAsEt hybrid system analyzer. As the picture shows, sAsEt builds on top of the Jahob system. The Jahob system provides a number of libraries for manipulating and proving high order formulae. In particular, it provides a number of rewriting techniques from higher order logic (HOL) to first order logic (FOL) theories. Once the formulae is rewritten in a suitable theory, Jahob provides interfaces for several theorem provers, e.g. Isabelle [108], Coq [29], or Spass [132]. For our experiments, we use Z3.

The two main components of the tool sAsEt are the automata and the analyzer libraries. The automata library contains definitions of timed automata, sequential timed automata, hybrid systems and the definitions of different transitions systems. The analyzer library implements important operations (e.g. parallel product) and our abstractions for timed automata and hybrid systems. The tool provides two parsers for reading and writing Uppaal timed automata or SpaceEx hybrid systems. Our tool supports the full syntax of SpaceEx models. Currently our tool supports a subset of the Uppaal syntax.

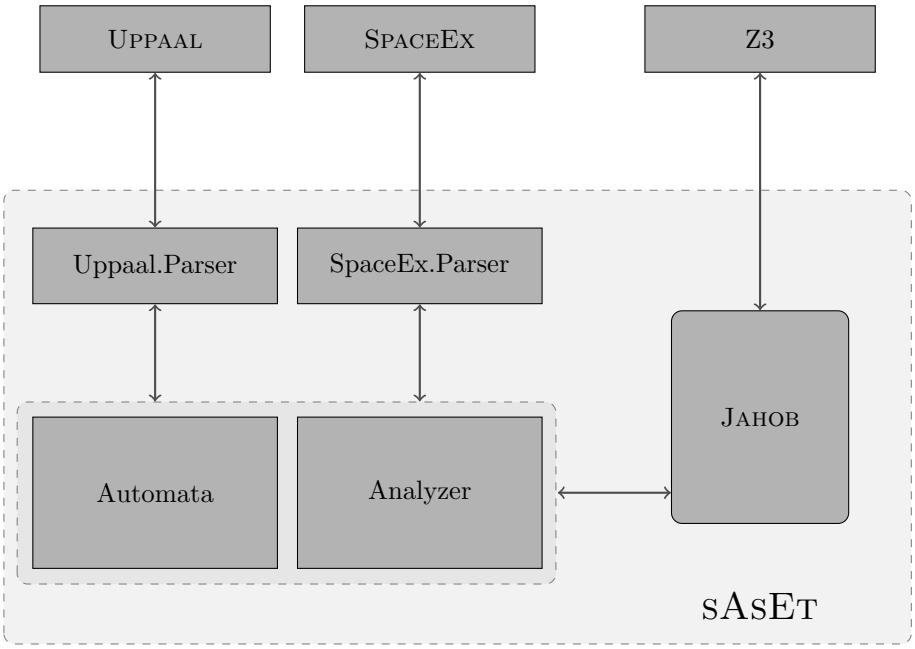


Figure 6.1: Architecture of the sAsEt system.

6.1.2 Implementation

Our tool is implemented in Ocaml 4 [117] under Linux [127]. We represent zones as first order linear real arithmetic formulae ($\mathsf{FOL}_{\mathsf{LRA}}$). We use SMT solvers for solving constraints and several libraries from Jahob to manipulate formulae and perform SMT calls.

Zones as Linear Real Arithmetic Predicates

We have implemented zones as linear real arithmetic ($\mathsf{FOL}_{\mathsf{LRA}}$). Formulae in sAsEt are HOL formulae as in the Isabelle theorem prover. Figure 6.2 shows a fragment of the HOL logic implemented in Jahob. Formulae are thus, variables, constants, application of a formula to a list of formula, a binder applied to identifiers in a formula, or a typed formula. For more details on formulae in the Jahob system we refer the reader to [90].

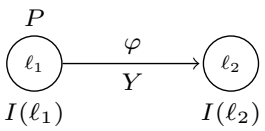
We now need to express the valuations of the automaton in terms of linear real arithmetic predicates. For this we use a strongest post condition operator. The definition of the strongest post condition operator is given in Figure 6.3 right. The strongest post condition operator takes as arguments a $\mathsf{FOL}_{\mathsf{LRA}}$ predicate and a clock constraint or a clock. The operator returns a $\mathsf{FOL}_{\mathsf{LRA}}$ predicate.

Figure 6.3 left presents a fragment of a timed automaton. The predicate P represents the clock valuations for some configuration $\langle \ell_1, P \rangle$. Given a predicate P and a clock constraint (guard or invariant) φ the post condition operator will intersect the predicate

```
1 type ident = string
2 type form =
3   | Var of ident
4   | Const of constValue
5   | App of form * form list
6   | Binder of binderKind * (typedIdent list) * form
7   | TypedForm of form * typeForm
```

Figure 6.2: Subset of the high order logic implemented in Jahob.

with the constraint. If a predicate P and a clock x are given, the operator will update the predicate P by setting the clock x equal to zero. The old value of the clock x is then existentially quantified. Sets of clock resets, are considered as a list of actions. The strongest post condition operator is inductively defined for lists of actions.



$$\mathsf{sp} : \mathsf{FOL}_{\mathsf{LRA}} \times (\Phi(\mathbb{X}) \cup \mathbb{X}) \to \mathsf{FOL}_{\mathsf{LRA}}$$

$$\begin{array}{lcl} \mathsf{sp}(P, \varphi) & \stackrel{\mathsf{def}}{=} & P \wedge \varphi \\ \mathsf{sp}(P, x) & \stackrel{\mathsf{def}}{=} & \exists x_0.\ x = 0 \wedge P[x/x_0] \\ \mathsf{sp}(P, s_1; \ldots; s_n) & \stackrel{\mathsf{def}}{=} & \mathsf{sp}(\mathsf{sp}(P, s_1), s_2; \ldots s_n) \end{array}$$

Figure 6.3: Left: a fragment of a timed automaton, the predicate P represents the clock valuations for some configuration $\langle \ell_1, P \rangle$. Right: definition of the strongest post condition operator. Given a predicate and a clock constraint or a clock, it computes the successor predicate.

Example 6.1. Consider the fragment of the automaton in Figure 6.3 left. Given the configuration $\langle \ell_1, P \rangle$ and $Y = \{x, y\}$. The symbolic successors are given by the $\mathsf{FOL}_{\mathsf{LRA}}$ predicate: $\mathsf{sp}(P, I(\ell_1); \varphi; x; y; I(\ell_2))$. ♢

Note that our definition of strongest post-condition do not consider delay transitions. Interestingly, for detecting quasi-equal clocks it is not necessary to compute delay successors. If a configuration is non zero-time, the relax operator is applied and the resulting zone consists of equalities among clocks. Equalities are closed with respect to the delay operator. If a configuration is zero-time meaning that the configuration do not have delay successors, then the configuration is normalized and stored.

Timed Automata

Figure 6.4 shows some the basic types of the automata library in the sAsEt tool. Since for our techniques the use of data types is irrelevant, our tool do not support data types such as integers. However, introducing data types does not change the spirit of our methods. Therefore, adding data types to our tool is straight forward.

The most basic type is the type channel, which are identifiers. The type actions is defined over channels. Actions are send, receive, or silent. The type location, represents a location as a list of identifiers. The type clock, represents a clock as variable of type real which is of type form. The type invariant is a formula. For efficiency the invariant function is implemented using a hash table which takes locations as keys. The type guard is a formula. The type edge is a tuple of locations, actions, a guard, a list of clocks and a location.

With these basic types we can define our type automaton. We define two types of automaton, timed automata as presented in our preliminary work in Chapter 2, and a sequential timed automata as presented in Chapter 4. As we see the definitions of both types are identical to the theoretical definitions. A timed automaton in sAsEt is a tuple consisting of a list of locations, a list of channels, a list of clocks, an invariant function, a list of edges and a final location. A sequential timed automaton in sAsEt is analogously defined.

6.1.3 A Session in the Timed Automata Analyzer

In the following we illustrate the use of our tool with an example. In the example, we use our tool to detect quasi-equal clocks. Our tool incorporates a number of operations on timed automata. Thus, it is a flexible framework for manipulating timed automata and formulae.

Example 6.2. Consider the automaton $\mathcal{A}$ in Figure 5.1. The automaton has clocks x and y. Figure 6.5 shows a session in the tool sAsEt. In this example, the tool takes as input a timed automata model and the command qeAnalysis which means that the input automata has to be analyzed for detecting quasi-equal clocks. The input automata is $\mathcal{A}$. Our tool provides a number of options for output visualization. Other options include, e.g. computing and storing the abstract zone graph.

The result of the analysis is an abstract zone graph and a binary relation stating which clocks are quasi-equal. In our example the output abstract transition system corresponds to the one in Figure 5.3. Note that, for zero time configurations, all the information is preserved. In particular, the value of the reset clocks is preserved by existentially quantified variables. ♢

```
1  type channel =
2    | Channel of ident
3    | Broadcast of ident
4
5  type action =
6    | Send of channel
7    | Receive of channel
8    | Tau of channel
9    | EmptyAction
10
11 type location = Location of ident list
12
13 type clock =  Clock of form
14
15 type invariant = Invariant of form InvariantHash.t
16
17 type guard = Guard of form
18
19 type edge =
20     Edge of location * action * guard * clock list * location
21
22 type automaton =
23     TimedAutomaton of
24         location list * channel list * clock list * invariant *
25           edge list * location
26     SequentialAutomaton of
27         location list * channel list * clock list * invariant *
28           edge list * location * location * int * int * int *
29             edge * clock
```

Figure 6.4: A subset of the data types in the tool sAsEt for representing timed automata. Basic types are identifiers which are strings, channels, actions over channels, locations, clocks, invariant function, guards as $\mathsf{FOL}_{\mathsf{LRA}}$ formulae, edges ,timed automata and sequential automata. The definition for hybrid systems is similar.

```
1
2  :~/sAsEt/bin$ ./sAsEt -qeAnalysis -modell chemicalPlant
3
4                    ======Abstract Zone Graph=========
5  configurations:
6   < l0 ,(EX (x_0_3::real).
7         ((x_0=0) & ((x_0_3=y_0) & (x_0_3<=60) & (y_0<=60) & (60<=x_0_3))))>
8   < l2 ,(EX (x_0_2::real).
9         ((x_0=0) & ((x_0_2=y_0) & (x_0_2<=60) & (y_0<=60) & (60<=x_0_2))))>
10  < l1 ,(EX (y_0_1::real).
11        ((y_0=0) & ((x_0=y_0_1) & (x_0<=60) & (y_0_1<=60) & (60<=y_0_1))))>
12  < l0 ,(EX (y_0_1::real).
13        ((y_0=0) & ((x_0=y_0_1) & (x_0<=60) & (y_0_1<=60) & (60<=y_0_1))))>
14  < l3 ,(x_0=y_0)>
15  < l2 ,(x_0=y_0)>
16  < l1 ,(x_0=y_0)>
17  < l0 ,(x_0=y_0)>
18
19 transitions:
20 < l0 ,(x_0=y_0)> ------> < l2 ,(x_0=y_0)>
21 < l0 ,(x_0=y_0)> ------> < l1 ,(x_0=y_0)>
22 < l1 ,(x_0=y_0)> ------> < l3 ,(x_0=y_0)>
23 < l2 ,(x_0=y_0)> ------> < l3 ,(x_0=y_0)>
24
25 < l1 ,(x_0=y_0)>
26 ------> < l0 ,(EX (x_0_3::real).
27        ((x_0=0) & ((x_0_3=y_0) & (x_0_3<=60) & (y_0<=60) & (60<=x_0_3))))>
28
29 < l2 ,(x_0=y_0)>
30 ------> < l0 ,(EX (y_0_1::real).
31        ((y_0=0) & ((x_0=y_0_1) & (x_0<=60) & (y_0_1<=60) & (60<=y_0_1))))>
32  ...
33  ...
34  ...
35 initial config:
36 < l0 ,(x_0=y_0)>
37
38 end of qe analysis with QE relation: (x_0,y_0)
39 time: 0.724058    QE configurations:8    smtCalls: 58
```

Figure 6.5: A session in the sAsEt hybrid system analyzer. The input automaton in the one in Figure 5.1 with quasi-equal clocks x and y. The output is the abstract zone graph depicted in Figure 5.3 and the relation QE. The clocks x and y are quasi-equal.

6.2 Detecting Quasi-equal Clocks

In this section, we present results for detecting quasi-equal clocks. For the detection of quasi-equal clocks we use our zero time abstraction method as presented in Chapter 5. In general our results show that the size of the transition system computed using our abstraction is very small in comparison to the one computed by UPPAAL. Note that UPPAAL computes a sound and complete abstraction using a normalization operator, e.g. the $\mathsf{norm}_{k,\mathcal{G}}$ operator presented in Section 2.1.2. Therefore, the verification times for SASET are fast in spite of the number of SMT calls which are time costly. We are thus not saying that our tool is faster than UPPAAL but we are showing that our abstraction is a useful one.

In Table 6.3, we present a number of results obtained by using our tool and the model checker UPPAAL [91]. Our intention is not to outperform UPPAAL in terms of time but to show that the abstraction method that we propose for detecting quasi-equalities is a good abstraction. Thus, we encourage the reader to focus on the number of states generated by the tools for each automaton. Note, that for UPPAAL to detect n clocks to be quasi-equal it would need to perform 2^n queries whereas our tool compute them directly as explained in Algorithm 5.1. In Table 6.3, max k is the number of the maximal constant appearing in the corresponding timed automaton and the queries φ are TCTL formulae asserting a number of clocks to be quasi-equal, e.g. $\varphi_1 := \mathsf{AG}\ x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0$, $\varphi_2 := \mathsf{AG}\ (x_0 = x_1 \vee x_0 = 0 \vee x_1 = 0) \wedge (x_0 = x_2 \vee x_0 = 0 \vee x_2 = 0)$. The experiments were executed in a AMD Phenom II X6 3.2Ghz Processor with 8GB RAM running Linux 3.2.

We use three classes of timed automata which are relevant for our abstraction. For classA the zero time behavior is constant and the non zero time behavior grows. For classB the zero time behavior grows linear and the non-zero time behavior remains constant. For classC, the zero time behavior grows exponentially.

The automata in classA correspond to the automaton in Figure 5.5. We only change the maximal constant appearing in the automaton and observe that the size of the abstraction remains the same. For the automata in classB the number of quasi-equal clocks is increased but there is an order on the reset of the quasi-equal clocks. We observe a linear increase in the number of states for both tools. For the automata in classC the number of quasi-equal clocks is increased but there is no order in the reset of the clocks. We observe an exponential increase in the number of states for both tools. In Table 6.3, the construction of the abstraction for automaton classC6 required 8515 SMT calls, according to our tool these SMT calls took 695.2 seconds, which is 82% of the time cost. Thus, a more efficient implementation is desirable. Since our algorithm can be implemented using difference bound matrices a much efficient implementation is possible. Figure 6.6 illustrates the results for automata in classC.

Once the quasi-equal clocks in a timed automaton have been detected. A reduction on the number of clocks might take place, leading to a major speed up. As an example we have reduced all the quasi-equal clocks for the automaton classC6 by replacing them with a representative clock. The resulting zone graph computed by UPPAAL consists of 5003 states and invariant properties can be verified in few seconds, which is an exponential gain.

Automaton	clocks	qe-clocks	max k	sAsEt			Uppaal		
				SMT-calls	states	t (s)	Q	states	t (s)
classA1	3	2	10^4	26	5	0.3	φ_1	20k	7.3
classA2	3	2	10^5	26	5	0.3	φ_1	200k	1200
classA3	3	2	10^6	26	5	0.3	φ_1	t.o.	t.o.
classB2	4	2	10^4	72	8	0.8	φ_1	20k	7.4
classB3	5	3	10^4	105	10	1.3	φ_2	30k	12.5
classB4	6	4	10^4	144	12	2.2	φ_3	40k	21.0
classB5	7	5	10^4	193	14	3.5	φ_4	50k	34.8
classB6	8	6	10^4	248	16	5.16	φ_5	60k	45.2
classC2	3	2	5000	63	7	1.2	φ_1	5k	5.2
classC3	4	3	5000	237	14	8.3	φ_2	35k	31.8
classC4	5	4	5000	809	26	44.05	φ_3	75k	202.3
classC5	6	5	5000	2389	47	195.3	φ_3	150k	1007.3
classC6	7	6	5000	8515	85	844	φ_4	t.o.	t.o.

Table 6.1: Results for detecting quasi-equal clocks using tools sAsEt and Uppaal. sAsEt returns the set of quasi-equal clocks whereas Uppaal performs a single query asserting clocks to be quasi-equal. Note, that for detecting quasi-equal clocks in Uppaal multiple queries are needed. The zero time behavior for automata in classA remains constant, in classB grows linearly and in classC grows exponentially.

6.2.1 Simplified Fire Alarm System

In this section, we apply our zero time abstraction for detecting quasi-equal clocks in the simplified fire alarm system (SFAS) described in Chapter 1, Section 1.3.

Table 6.2 presents the results of our experiments. We present two different experiments. The first one consist in computing the parallel product on a number of sensors and then running the analysis (since the SFAS has 125 sensors this approach does not scale). The second approach is an incremental approach consisting in a pairwise comparison of quasi-equal clocks. Since the quasi-equal relation is an equivalence relation, the relation is transitive. Therefore, in order to detect all quasi-equal clocks, in the worst case we need to run the analysis $\mathcal{O}(n^2)$ times, where n is the number of components.

In Table 6.2. The column automaton, describes the model being analyzed. We use the acronym SFAS_i for simplified fire alarm system, where i indicates the number of sensors starting from 0. For the incremental approach SFAS_i_j indicates the two components i and j that are being analyzed. The column qe-clocks, describes the set of quasi-equal clocks detected. The column sAsEt, describes the number of SMT-calls,

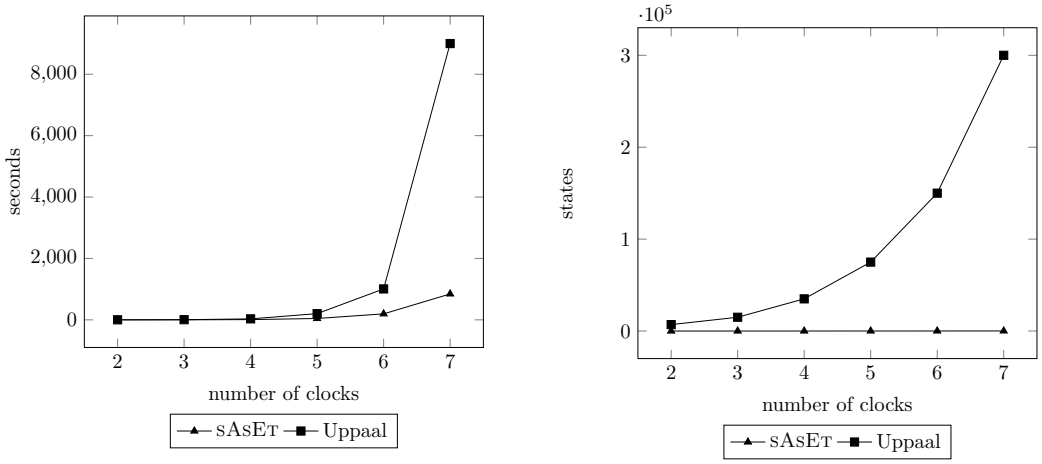


Figure 6.6: Left: Verification times for detecting quasi-equal clocks for automata in classC. Right: Number of states explored for detecting quasi-equal clocks for automata in classC. Note that in contrast to classA and classB the number of states in sAsEt and Uppaal grow exponentially. However, the number of states in sAsEt is much smaller.

the number of generated states and the time required to detect the quasi-equal clocks.

For the first experiment, the input network SFAS_i consisting of i components is given. We observe that this approach do not scale. As soon as there are 7 clocks, the system times out. This is because, there is a diamond structure corresponding to all the possible reset permutations of the 7 clocks. Since this diamond is zero time, the explosion occurs in both our tool sAsEt and in Uppaal.

Since the simplified fire alarm system consist of 125 sensors, our first attempt do not scale. In our second experiment we use a different strategy. Our strategy exploits the fact that the quasi-equal relation is an equivalence relation. Therefore, because of transitivity we can perform a piecewise analysis on the 125 sensors. In the worst case this incremental approach will need to be run $\mathcal{O}(n^2)$ times, where n is the number of components. In the case of the fire alarm system we were able to detect all quasi-equal clocks in 79.8 seconds.

Once quasi-equal clocks have been detected. A reduction on the number of clocks can take place. By using the techniques presented in [76], a reduction on the number of quasi-equal clocks for the simplified fire alarm system will reduce the verification times of the system from exponential to quadratic in the number of components. Further, by using the techniques presented in Chapter 4, the verification times for the simplified fire alarm system will reduce from exponential to linear in the number of components.

Automaton	QE-clocks	sAsEt		
		SMT-calls	states	t (s)
SFAS_1	x_0, x_1	38	9	0.66
SFAS_2	x_0, x_1, x_2	93	16	2.43
SFAS_3	x_0, x_1, x_2, x_3	210	27	7.74
SFAS_4	x_0, x_1, x_2, x_3, x_4	465	46	35.04
SFAS_5	$x_0, x_1, x_2, x_3, x_4, x_5$	1030	81	388.23
SFAS_6	$x_0, x_1, x_2, x_3, x_4, x_5, x_6$	T.O.	T.O.	T.O.
SFAS_0_1	x_0, x_1	38	9	0.66
SFAS_1_2	x_1, x_2	38	9	0.64
SFAS_2_3	x_2, x_3	38	9	0.66
SFAS_3_4	x_3, x_4	38	9	0.63
⋮	⋮	⋮	⋮	⋮
SFAS_123_124	x_{123}, x_{124}	38	9	0.66
		$\sum = 4712$		$\sum = 79.8$

Table 6.2: Results for detection of quasi-equal clocks in the simplified fire alarm system (SFAS). Top: analysis of quasi-equal clocks on the network SFAS_i, where i is the number of components. The approach times out with 7 components. Bottom: incremental analysis of quasi-equal clocks in network SFAS_i_j with components i and j. This analysis (pairwise comparison) is possible because the quasi-equal relation is a transitive relation.

6.3 Real World Fire Alarm System

In Chapter 1, Section 1.3 we have introduced a simplified version of a real world fire alarm system. We now show with experiments the effect of applying our techniques to the corresponding real world fire alarm system. We successfully reduce the verification times from exponential to linear in the number of components.

We consider a real world fire alarm system which we denote by *FAS* (the system is being developed by a German company; an anonymized version of a model of the system has been made public and can be found in [58]). The *FAS* monitors n sensors using m channels. In order, for *FAS* to obtain an EU quality certificate it has to be conform, among others, with the following condition: If a sensor is malfunctioning, the failure has to be recognized in less than 300 seconds. We denote this property by AG *less300*.

System	Q	Broadcast		Sequential	
		t (s)	states	t (s)	states
FAS-CB-SW	φ_1	1103.9	5947.4k	318.1	5971.5k
	φ_2	2240.5	11508.3k	704.2	11614.5k
FAS-CB	φ_1	196.4	1184.7k	120.3	1189.5k
	φ_2	272.6	165.7k	150.3	1666.9k
FAS-SW	φ_1	13.7	104.1k	87.4	104.7k
	φ_2	10.62k	145.5	87.4	146.4k
FAS	φ_1	2.5	20.7k	87.5	20.8k
	φ_2	1.3	20.7k	85.6	20.8k

Table 6.3: Verification times (AMD Opteron 6174 2.2GHz, 64Gb RAM) and visited states for satisfied properties $\varphi_1 = (\mathsf{AG}\ \textit{not deadlock})$ and $\varphi_2 = (\mathsf{AG}\ \textit{less300})$ for the fire alarm system with 125 sensors using UPPAAL,

In addition, the certifying institution is able to: (i) block an arbitrary channel for any number of seconds, then (ii) release the blocked channel for at least 1 second and repeat (i), and (iii) execute (ii) any number of times.

In order to model the above mentioned situations, we constructed a sensor switcher *SW* which non-deterministically turns off any sensor. We constructed a channel blocker *CB*, which models the blocking of channels as described above. Now, let *FAS-CB* denote the fire alarm system together with the channel blocker *CB*. Let *FAS-SW* be *FAS* together with the sensor switcher. Let *FAS-CB-SW* be *FAS* together with the channel blocker and a sensor switcher.

Table 6.3, show the verification results for the satisfied properties AG *not deadlock* and AG *less300* for the corresponding system with 125 sensors by using UPPAAL. The times include the parsing time of UPPAAL templates. The attempt of modeling a sensor, with its own clock, did not scale to more than 10 sensors. Therefore, our modelers optimized the models by reducing the quasi-equal clocks using the approach presented in [76]. Thus, all 125 sensors share one clock and synchronizations are performed via a broadcast channel. This optimized models correspond to the column Broadcast. Models corresponding to the column Sequential are constructed using the sequential composition operator presented in Chapter 4.

We observe that for a large state space, sequential is much faster that broadcast. In fact a quadratic speed up is achieved (non-enabled edges are not present in sequential). This improvement is justified by Lemma 3.22. However, for small state space such as *FAS* broadcast is faster; in this context, note that the parsing time for the large template consisting of 125 sequentialized automata is taking about 85 sec. That is, the difference

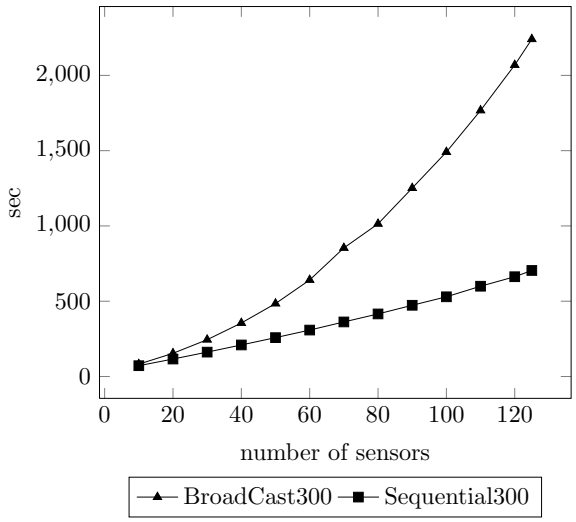


Figure 6.7: Verification times of *FAS-CB-SW* for property (AG *less300*) over number of sensors compared to the corresponding fired alarm system obtained by sequential composition.

is due to the parsing of a big timed automata versus the parsing of a network of small automata in Uppaal.

Figure 6.7 depicts the verification times of the system *FAS-CB-SW* and property AG *less300* for 10, 20, ..., 120, and finally 125 sensors, the curve for broadcast is comparable with the statement of Lemma 3.22. That is, there is a quadratic unnecessary number of enablelesness of edges. The removal of this unnecessary edges by using the sequential composition operator yields a quadratic speed ups (cf. Figure 6.7).

6.4 Future Work

As a proof of concept we have implemented our tool sAsEt for performing analysis in hybrid systems. Due to the abstract and modular structure of our tool. It is easy to further implement other analysis, transformations or encodings for hybrid systems. In the following we summarize important aspects which could be further implemented in our tool for increasing its usability.

- Extend the tool to a full model checker for timed automata, i.e. compute a finite, complete and sound abstraction using a normalization operator e.g. the $\mathsf{norm}_{k,\mathcal{G}}$ operator.

- Our tool uses SMT solvers for discharging constraints. This enables us to solve not only difference logic constraints but more complex constraints which arise during

the analysis of hybrid systems. However, for the case of timed automata, adding support for the DBM library will significantly increase the performance of our tool.

- Since the syntax of UPPAAL is widely accepted, the implementation of an interpreter for UPPAAL expressions would enable us to fully work with UPPAAL models.
- Since our tool is implemented in top of the JAHOB system. Our tool performs a number of optimizations for discharging constraints, e.g. syntactic checking for validity of formulae, avoiding the use of SMT solvers when possible. When analyzing hybrid systems, a number of existentially quantified variables arise. Therefore, an implementation of a quantifier elimination procedure for $\mathsf{FOL}_{\mathsf{LRA}}$is desirable.

Chapter 7

Conclusion

In this thesis we have studied the formal verification of Time Division Multiple Access (TDMA) based systems. TDMA based systems, are used to enable the communication of several components using a shared communication channel. The TDMA paradigm avoids message collisions using time division multiplexing. Since TDMA systems use dense time, the theory of timed automata is natural formal model to describe these systems.

The theory of timed automata, has been successfully used to formally verify several real world applications. The theory of timed automata is subject of several publications and is an area of active research. Despite of much effort from the research community, the formal verification using timed automata, is often hindered by the so called state explosion problem. The state explosion problem refers to the issue that given a number of timed automata, the number of states of the model grows exponentially on the number of automata and on the number of clocks. Since TDMA based systems include a large number of components and clocks, the state explosion problem is an issue for the formal verification of TDMA based systems.

In this thesis, we concern to the applicability of the theory of timed automata to formally verify TDMA based systems. Towards this goal, we present a number of techniques for avoiding the state explosion problem and to speed up the verification times.

In Chapter 3, we have formalized important characteristics of TDMA based systems. We introduce concepts such as periodic cyclic timed automata, disjoint activity, sequentialialisability and a concatenation operator. Given a number of sequentialisable periodic cyclic timed automata. The application of the concatenation operator will produce a system which is bisimilar(even identical) to the one obtained by using the parallel product operator. However, there is a substantial reduction on the number of locations and edges. This leads to better verification times.

In Chapter 4, we further concern with the applicability of timed automata to the formal verification of TDMA based systems. Since proving semantic properties (e.g. sequentialisability) might be as hard as the model checking task. We propose a number of syntactic patterns on which relevant semantically properties (e.g. sequentialisability) can be syntactically check. We call this class of syntactic automata, sequential timed automata. For sequentialisable sequential timed automata, we provide a sequential composition operator. The sequential composition operator yields a much smaller system which is weak bisimilar to the corresponding one obtained by parallel composition. Using

the patterns, will render the time complexity from exponential to linear on the number of components.

In Chapter 5, we consider a phenomena which is not only relevant to TDMA based systems, but to the complete class of timed systems. The phenomena is because of the semantics of timed automata (and other timed models). For equal clocks and a reset sequence of transitions, clocks may reset their values in zero time. This may cause clocks to be un-equal for intervals of time of zero duration. At the end of the reset sequence clocks will be equal again. We call this clocks quasi-equal. A reduction on the number of quasi-equal clocks has been proven to be an effective technique. Existing reduction techniques take as input the set of quasi-equal clocks. We introduce an abstraction for efficiently detecting quasi-equal clocks. Thus improving the verification of timed automata with quasi-equal clocks.

In Chapter 6, we present our proof of concept. We present our tool sAsEt for analysis in hybrid systems. The we present a number of experiments using the techniques described in this thesis. Our results are encouraging and ratify the applicability of our approaches. In particular, we show the results of verifying a real world TDMA based system. By using our techniques, the verification times from the real world system dropped from exponential in the number of components to linear in the number of components.

Because of our results. We believe, that the techniques presented in this thesis, are relevant and of practical impact for formally verifying TDMA based systems. Therefore, we contribute to the formal verification of TDMA based systems using timed automata as formal models. By using the syntactical patterns given in Chapter 4. The TDMA components can be modeled and relevant properties can be verified. In our experience, this is the fastest method for this class of systems (linear complexity in the number of components). The techniques in Chapter 5 enable the reduction on the number of clocks in timed automata. As a result, the corresponding models of, TDMA based systems and several safety critical systems. Can be simplified and efficiently verified.

7.1 Future Work

We would like to conclude this thesis with an outlook on possible directions for future work.

A More General Sequential Operator The sequential composition operator from Chapter 4 relies on disjoint activity from the components, the disjoint activity is determined by points of time. A possible extension might consist on the use of tokens. Tokens could be used for indicating the ending of the activity of one component. When a component is done with its activity the component could pass the token to the next component. Tokens will ensure disjoint activity, and would enable the efficient verification of other multiplexing protocols.

Activity Analysis and Smart Operators

In the case of TDMA based systems. The architecture of the paradigm and the time slots, give us intervals of time with disjoint activity. The concatenation operator and the sequential composition operator from Chapter 3 and from Chapter 4 respectively, exploit this information to construct a more compact system. The resulting system preserves all or most of its properties. Since at points of time where two components have disjoint activity, only one component is able to perform actions, computing the parallel composition for these points of time is unnecessary and costly. This fact lead us to think on analysis for detecting activity phases and on smart operators which may profit from the disjoint activity.

Generalize Quasi-equal Clocks

In Chapter 5, quasi-equal clocks were presented. Quasi-equal clocks are always equal except at some points of time where the clocks are reset. In [30], we have generalized the notion of quasi-equal clocks to the notion of quasi-dependent variables for hybrid systems. Quasi-dependent variables are always related via a function, except at some points of time, where the variables can have arbitrary values. One possible extension would be to allow a bounded difference value for variable valuations with respect to a master variable. We can define and equivalence relation among these variables and simplify them. This will allow to model systems with clock drifting while the simplified system will be tractable.

Local Properties

Since TDMA systems consist of a large number of components, the corresponding models are often untractable. However, in the case of TDMA based systems, many properties may refer to only one component. That is, many properties can be locally proven in a given component. Our idea is to construct only the relevant part of the transition system to correctly proof a given property. The construction of the transition system can be done by the semantics and a parallel composition operator at the transition system level. We believe, that in many cases the relevant transition system would be much smaller.

Chapter 8

Zusammenfassung

Software und Hardware Systeme sind allgegenwärtig. Diese Systeme steuern, e.g. Flugzeuge, Satelliten, Herzschrittmacher, etc. Eine große und besondere Klasse von Systemen, ist die Klasse, die ein Zeitmultiplexverfahren (Time Division Multiple Access TDMA) Protokoll zur Kommunikation nutzt. Die Anwesenheit von Fehlern oder unerwartetem Verhalten in solchen Systemen, kann zu schweren Konsequenzen führen, e.g. den Tod von Menschen, katastrophale wirtschaftliche Verluste, etc. Daher ist das korrekte Funktionieren von solchen Systemen kritisch.

Die Sicherstellung eines fehlerfreien Systems ist heute eines der aktivsten Forschungsgebiete. Ein erfolgversprechendes Verfahren zur vollautomatischen Verifikation einer Systembeschreibung ist die Modellprüfung (model checking) mit Zeitautomaten (timed automata). Zeitautomaten sind zur Modellprüfung von industriellen Systemen erfolgreich angewendet worden. Allerdings wird in besonderen Fällen der induzierte Zustandsraum unpraktikabel groß. Folglich gibt es in der Modellprüfungsgemeinde zahlreiche Publikationen, die sich mit Verfahren, um den Zustandsraum zu verkleinern, beschäftigen.

In dieser Dissertation tragen wir zur Modellprüfung von TDMA Systemen mit Zeitautomaten bei. Wir untersuchen die Prinzipien von TDMA Systemen. Dann charakterisieren wir diese Prinzipien in der Zeitautomatentheorie. Im Allgemeinen stellen wir verschiedene Verfahren vor. Das Ziel unserer Verfahren ist den Zustandsraum zu reduzieren und alle oder die meisten Eigenschaften zu bewahren.

In dieser Arbeit geben wir als Erstes eine semantische Formalisierung der Kernprinzipien von TDMA Systemen. Dazu führen wir die Definitionen disjunkt Aktivität, Sequentialisierbarkeit und Konkatenierung ein. Wir beachten, dass die Anwendung dieser Definitionen zu einer quadratischen Beschleunigung führen kann. Zweitens fokussieren wir uns auf die Anwendbarkeit von Echtzeitmodellprüfung. Zu diesem Zweck präsentieren wir syntaktische Vorlagen und Operatoren zur Modellierung von TDMA Systemen. Für TDMA Systeme, die mit den syntaktischen Vorlagen modelliert wurden, haben wir eine Reduktion in der Zeitkomplexität von exponentiell zu linear erzielt. Schließlich erkennen wir einen weiteren wichtigen Bestandteil von Echtzeitsystemen, i.e. Nullzeit-Uhren-Differenzen. Die Differenzen entstehen als semantische Konsequenz der Zeitautomatentheorie. Wir nennen diese Uhren quasi-equal Uhren. Eine Reduktion in der Anzahl von Uhren führt zu einem reduzierten System. Bevor eine Reduktion stattfinden kann, müssen die quasi-equal Uhren effizient erkannt werden. Daher präsentieren wir ein effizientes Abstraktionsverfahren, um quasi-equal Uhren zu erkennen.

Als Konzeptnachweis haben wir unsere Methoden in unserem Tool SAsEt implementiert. Unsere Experimente bestätigen und motivieren die Anwendung von den in dieser Thesis vorgestellten Verfahren. Zusätzlich wurden die oben genannten Verfahren zur Verifizierung eines industriellen TDMA Systems erfolgreich angewendet.

Bibliography

[1] Alur, R.: Formal verification of hybrid systems. In: Embedded Software (EMSOFT), 2011 Proceedings of the International Conference on. pp. 273–278. IEEE (2011)

[2] Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. 104(1), 2–34 (May 1993)

[3] Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A., Ho, P.H., Nicollin, X., Olivero, A., Sifakis, J., Yovine, S.: The algorithmic analysis of hybrid systems. Theoretical computer science 138(1), 3–34 (1995)

[4] Alur, R., Dang, T., Ivancic, F.: Reachability analysis of hybrid systems via predicate abstraction. In: Tomlin, C., Greenstreet, M. (eds.) Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 2289, pp. 35–48. Springer Berlin Heidelberg (2002)

[5] Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science 126(2), 183–235 (1994)

[6] Alur, R., Fix, L., Henzinger, T.A.: A determinizable class of timed automata. In: Computer Aided Verification. pp. 1–13. Springer-Verlag (1994)

[7] Alur, R., Henzinger, T.A., Vardi, M.Y.: Parametric real-time reasoning. In: Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing. pp. 592–601. STOC '93, ACM, New York, NY, USA (1993)

[8] Asarin, E., Caspi, P., Maler, O.: A kleene theorem for timed automata. In: Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science. pp. 160–. LICS '97, IEEE Computer Society, Washington, DC, USA (1997)

[9] Baader, F., Nipkow, T.: Term rewriting and all that. Cambridge University Press (1999)

[10] Baier, C., Katoen, J.P.: Principles of Model Checking (Representation and Mind Series). The MIT Press (2008)

[11] Ball, T., Majumdar, R., Millstein, T., Rajamani, S.K.: Automatic predicate abstraction of c programs. In: ACM SIGPLAN Notices. vol. 36, pp. 203–213. ACM (2001)

[12] Barrett, C., Berezin, S.: CVC Lite: A new implementation of the cooperating validity checker. In: CAV. LNCS, vol. 3114 (2004)

[13] Barrett, C., Stump, A., Tinelli, C.: The smt-lib standard: Version 2.0. In: Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, England). vol. 13, p. 14 (2010)

[14] Barrett, C., Tinelli, C.: CVC3. In: CAV. LNCS, vol. 4590 (2007)

[15] Behrmann, G., Bengtsson, J., David, A., Larsen, K.G., Pettersson, P., Yi, W.: Uppaal implementation secrets. In: Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 3–22. Springer-Verlag (2002)

[16] Behrmann, G., Bouyer, P., Fleury, E., Larsen, K.G.: Static guard analysis in timed automata verification. In: Proceedings of the 9th international conference on Tools and algorithms for the construction and analysis of systems. pp. 254–270. TACAS'03, Springer-Verlag, Berlin, Heidelberg (2003)

[17] Behrmann, G., Bouyer, P., Larsen, K., Pelánek, R.: Lower and upper bounds in zone based abstractions of timed automata. Tools and Algorithms for the Construction and Analysis of Systems pp. 312–326 (2004)

[18] Behrmann, G., Bouyer, P., Larsen, K.G., Pelánek, R.: Lower and upper bounds in zone-based abstractions of timed automata. International Journal on Software Tools for Technology Transfer 8(3), 204–215 (2006)

[19] Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: Uppaal-tiga: Time for playing games! In: Computer Aided Verification. pp. 121–125. Springer-Verlag (2007)

[20] Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. pp. 200–236. No. 3185 in LNCS, Springer-Verlag (2004)

[21] Bellman, R., Kalaba, R.E.: Dynamic programming and modern control theory. Academic Press New York (1965)

[22] Ben-Ari, M.: Mathematical Logic for Computer Science. Series in Computer Science, Prentice Hall International (1993)

[23] Ben Salah, R., Bozga, M.D., Maler, O.: Compositional timing analysis. In: Proceedings of the Seventh ACM International Conference on Embedded Software. pp. 39–48. EMSOFT '09, ACM, New York, NY, USA (2009)

[24] Bengtsson, J.: Clocks, dbms and states in timed systems. Acta Universitatis Upsaliensis (2002)

[25] Bengtsson, J., Jonsson, B., Lilius, J., Yi, W.: Partial order reductions for timed systems. In: CONCUR'98 Concurrency Theory, pp. 485–500. Springer-Verlag (1998)

[26] Bengtsson, J., Yi, W.: On clock difference constraints and termination in reachability analysis of timed automata. Formal Methods and Software Engineering pp. 491–503 (2003)

[27] Bengtsson, J., Yi, W.: Timed automata: Semantics, algorithms and tools. In: Advances in Petri Nets. pp. 87–124. Springer-Verlag (2004)

[28] Bensalem, S., Lakhnech, Y., Owre, S.: Computing abstractions of infinite state systems compositionally and automatically. In: Computer Aided Verification. pp. 319–331. Springer-Verlag (1998)

[29] Bertot, Y., Castéran, P.: Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions. springer (2004)

[30] Bogomolov, S., Herrera, C., Muñiz, M., Westphal, B., Podelski: Quasi-dependent variables in hybrid automata. In: HSCC2014. LNCS, Springer Berlin Heidelberg (2014)

[31] Börger, E., Grädel, E., Gurevich, Y.: The classical decision problem. Springer-Verlag (2001)

[32] Bouyer, P.: Untameable timed automata! In: STACS 2003, pp. 620–631. Springer-Verlag (2003)

[33] Bouyer, P., Petit, A.: Decomposition and composition of timed automata. In: Proceedings of the 26th International Colloquium on Automata, Languages and Programming. pp. 210–219. ICAL '99, Springer-Verlag, London, UK (1999)

[34] Bradley, A.R., Manna, Z.: The Calculus of Computation. Springer-Verlag (2007)

[35] Bratley, P., Fox, B.L., Schrage, L.E.: A guide to simulation, vol. 2. Springer-Verlag (1983)

[36] de Bruijn, N.G.: Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. Indag. Math. 34, 381–392 (1972)

[37] Chaki, S., Clarke, E., Groce, A., Jha, S., Veith, H.: Modular verification of software components in C. TSE 30(6) (2003)

[38] Chaochen, Z., Ravn, A.P., Hansen, M.R.: An extended duration calculus for hybrid real-time systems. Springer-Verlag (1993)

[39] Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Computer Aided Verification. pp. 258–263. Springer-Verlag (2013)

[40] Cimatti, A., Palopoli, L., Ramadian, Y.: Symbolic computation of schedulability regions using parametric timed automata. In: Real-Time Systems Symposium, 2008. pp. 80–89 (Nov 2008)

[41] Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems (TOPLAS) 8(2), 244–263 (1986)

[42] Clarke, E.M., Kurshan, R.: Computer-aided verification. Spectrum, IEEE 33(6), 61–67 (1996)

[43] Clarke, E.M., Schlingloff, B.H.: Model checking. In: Robinson, A., Voronkov, A. (eds.) Handbook of Automated Reasoning, pp. 1635–1790. Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands (2001)

[44] Clarke, E.M., Wing, J.M.: Formal methods: State of the art and future directions. ACM Computing Surveys (CSUR) 28(4), 626–643 (1996)

[45] Clarke, E., Emerson, E.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking, Lecture Notes in Computer Science, vol. 5000, pp. 196–215. Springer Berlin Heidelberg (2008)

[46] Cook, B., Podelski, A. (eds.): Verification, Model Checking, and Abstract Interpretation, 8th International Conference, VMCAI 2007, Nice, France, January 14-16, 2007, Proceedings, Lecture Notes in Computer Science, vol. 4349. Springer-Verlag (2007)

[47] Cousot, P.: Verification by abstract interpretation. In: Dershowitz, N. (ed.) Proc. Int. Symp. on Verification – Theory & Practice – Honoring Zohar Manna's 64th Birthday. pp. 243–268. © Springer-Verlag, Berlin, Germany, Taormina, Italy (June 29 – July 4 2003)

[48] Cousot, P., Cousot, R.: Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. 4th POPL (1977)

[49] Cousot, P., Cousot, R.: Abstract interpretation frameworks. JLC 2(4), 511–547 (Aug 1992)

[50] Cousot, P., Cousot, R.: Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. In: Lecture Notes in Computer Science. pp. 269–295 (1992)

[51] Dalsgaard, A.E., Laarman, A., Larsen, K.G., Olesen, M.C., Van De Pol, J.: Multi-core reachability for timed automata. In: Formal Modeling and Analysis of Timed Systems, pp. 91–106. Springer-Verlag (2012)

[52] Dams, D., Gerth, R., Knaack, B., Kuiper, R.: Partial-order reduction techniques for real-time model checking. Formal Aspects of Computing 10(5-6), 469–482 (1998)

[53] David, A.: Uppaal dbm library programmer's reference (2006)

[54] Daws, C., Yovine, S.: Reducing the number of clock variables of timed automata. In: Proc. RTSS'96, 73-81, IEEE. pp. 73–81. IEEE Computer Society Press (1996)

[55] Daws, C., Tripakis, S.: Model checking of real-time reachability properties using abstractions. In: Proceedings of the 4th International Conference on Tools and Algorithms for Construction and Analysis of Systems. pp. 313–329. Springer-Verlag, London, UK (1998)

[56] De Moura, L., Bjørner, N.: Z3: An efficient smt solver. Tools and Algorithms for the Construction and Analysis of Systems pp. 337–340 (2008)

[57] Demichelis, F., Zielonka, W.: Controlled timed automata. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR'98 Concurrency Theory, Lecture Notes in Computer Science, vol. 1466, pp. 455–469. Springer Berlin Heidelberg (1998)

[58] Dietsch, D., Feo-Arenis, S., Muñiz, M., Andisha, A.S., Westphal, B.: The wireless fire alarm system: Ensuring conformance to industrial standards through formal verification. In: FM2014. LNCS, Springer Berlin Heidelberg (2014)

[59] Dietsch, D., Feo-Arenis, S., Westphal, B., Podelski, A.: An accountable approach to formal methods in small companies (2011), submitted for publication

[60] Dietsch, D., Feo-Arenis, S., Westphal, B., Podelski, A.: Disambiguation of industrial standards through formalization and graphical languages. In: IEEE 19th International Requirements Engineering Conference. pp. 265–270 (2011)

[61] Dill, D.L.: Timing assumptions and verification of finite-state concurrent systems. In: Automatic verification methods for finite state systems. pp. 197–212. Springer-Verlag (1990)

[62] Elmenreich, W., Ipp, R.: Introduction to ttp/c and ttp/a. na (2003)

[63] Elrad, T., Francez, N.: Decomposition of distributed programs into communication-closed layers. Sci. Comput. Program. 2(3), 155–173 (1982)

[64] Engels, H.: CAN-bus. Franzis (2000)

[65] Flanagan, C., Qadeer, S.: Predicate abstraction for software verification. In: ACM SIGPLAN Notices. vol. 37, pp. 191–202. ACM (2002)

[66] Flanagan, C., Saxe, J.B.: Avoiding exponential explosion: Generating compact verification conditions. In: POPL01 (2001)

[67] Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: Spaceex: Scalable verification of hybrid systems. In: Computer Aided Verification. pp. 379–395. Springer-Verlag (2011)

[68] Gagliardi, R.: Time-division multiple access. In: Satellite Communications, pp. 251–288. Springer Netherlands (1991)

[69] Graf, S., Saidi, H.: Construction of abstract state graphs with pvs. In: Grumberg, O. (ed.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 1254, pp. 72–83. Springer Berlin Heidelberg (1997)

[70] Gruhn, V., Laue, R.: Patterns for timed property specifications. Electronic Notes in Theoretical Computer Science 153(2), 117 – 133 (2006), proceedings of the Third Workshop on Quantitative Aspects of Programming Languages (QAPL 2005) Quantitative Aspects of Programming Languages 2005

[71] Haartsen, J.C.: The bluetooth radio system. Personal Communications, IEEE 7(1), 28 –36 (feb 2000)

[72] Hakansson, J., Pettersson, P.: Partial order reduction for verification of real-time components. In: Raskin, J.F., Thiagarajan, P. (eds.) Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol. 4763, pp. 211–226. Springer Berlin Heidelberg (2007)

[73] Harper, R., Honsell, F., Plotkin, G.: A framework for defining logics. Journal of the ACM (JACM) 40(1), 143–184 (1993)

[74] Heiner, G., Thurner, T.: Time-triggered architecture for safety-related distributed real-time systems in transportation systems. In: FTCS. pp. 402–407 (1998)

[75] Henzinger, T., Nicollin, X., Sifakis, J., Yovine, S.: Symbolic model checking for real-time systems. Information and Computation 111(2), 193 – 244 (1994)

[76] Herrera, C., Westphal, B., Feo-Arenis, S., Muñiz, M., Podelski, A.: Reducing quasi-equal clocks in networks of timed automata. In: FORMATS. pp. 155–170. No. 7595 in LNCS, Springer-Verlag (2012)

[77] Herrera, C., Westphal, B., Podelski, A.: Quasi-equal clock reduction: More networks, more queries. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 295–309. Springer-Verlag (2014)

[78] Hindley, R.: The principal type-scheme of an object in combinatory logic. Transactions of the american mathematical society pp. 29–60 (1969)

[79] Hoenicke, J., Meyer, R., Olderog, E.R.: Kleene, rabin, and scott are available. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010 - Concurrency Theory (CONCUR). Lecture Notes in Computer Science, vol. 6269, pp. 462–477. Springer-Verlag (2010)

[80] Hoenicke, J.: Combination of Processes, Data, and Time. Ph.D. thesis, University of Oldenburg (July 2006)

[81] Hoenicke, J., Olderog, E.R.: Csp-oz-dc: A combination of specification techniques for processes, data and time. Nord. J. Comput. 9(4), 301–334 (2002)

[82] Holzmann, G.J.: Design and validation of protocols: a tutorial. Computer Networks and ISDN Systems 25(9), 981–1017 (1993)

[83] Immerman, N.: Descriptive Complexity. Springer-Verlag (1998)

[84] Immerman, N., Rabinovich, A.M., Reps, T.W., Sagiv, S., Yorsh, G.: The boundary between decidability and undecidability for transitive-closure logics. In: Computer Science Logic (CSL). pp. 160–174 (2004)

[85] Jacobs, S.: Incremental instance generation in local reasoning. In: CAV. pp. 368–382 (2009)

[86] Janssen, W., Poel, M., Xu, Q., Zwiers, J.: Layering of real-time distributed processes. In: Proceedings of the Third International Symposium Organized Jointly with the Working Group Provably Correct Systems on Formal Techniques in Real-Time and Fault-Tolerant Systems. pp. 393–417. ProCoS, Springer-Verlag, London, UK, UK (1994)

[87] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang: Symbolic Model Checking: 10^{20} States and Beyond. In: Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science. pp. 1–33. IEEE Computer Society Press, Washington, D.C. (1990), `citeseer.ist.psu.edu/burch90symbolic.html`

[88] Jubran, O., Westphal, B.: Formal approach to guard time optimization for tdma. In: Proceedings of the 21st International conference on Real-Time Networks and Systems. pp. 223–233. ACM (2013)

[89] Kopetz, H., Grünsteidl, G.: Ttp - a time-triggered protocol for fault-tolerant real-time systems. In: FTCS. pp. 524–533 (1993)

[90] Kuncak, V.: Modular data structure verification. Ph.D. thesis, Massachusetts Institute of Technology (2007)

[91] Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 134–152 (1997)

[92] Larsen, K.G., Pettersson, P., Yi, W.: Uppaal: Status & developments. In: Computer Aided Verification. pp. 456–459. Springer-Verlag (1997)

[93] Law, A.M., Kelton, W.D., Kelton, W.D.: Simulation modeling and analysis, vol. 2. McGraw-Hill New York (1991)

[94] Mahfoudh, M., Niebert, P., Asarin, E., Maler, O.: A satisfiability checker for difference logic. Proceedings of SAT 2, 222–230 (2002)

[95] Makowitz, R., Temple, C.: Flexray-a communication network for automotive control systems. In: 2006 IEEE International Workshop on Factory Communication Systems. pp. 207–212 (2006)

[96] Merz, S.: Model checking: A tutorial overview. In: Modeling and verification of parallel processes, pp. 3–38. Springer-Verlag (2001)

[97] Miller, C., Scholl, C., Becker, B.: Verifying incomplete networks of timed automata. In: MBMV. pp. 113–122 (2011)

[98] Minea, M.: Partial order reduction for model checking of timed automata. Springer-Verlag (1999)

[99] Morbé, G., Pigorsch, F., Scholl, C.: Fully symbolic model checking for timed automata. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 6806, pp. 616–632. Springer Berlin Heidelberg (2011)

[100] Muñiz, M., Westphal, B., Podelski, A.: Timed automata with disjoint activity. In: FORMATS. pp. 188–203. No. 7595 in LNCS, Springer-Verlag (2012)

[101] Muñiz, M., Westphal, B., Podelski, A.: Detecting quasi-equal clocks in timed automata. In: Formal Modeling and Analysis of Timed Systems, pp. 198–212. Springer-Verlag (2013)

[102] Myers, G.J., Sandler, C., Badgett, T.: The art of software testing. John Wiley & Sons (2011)

[103] Navabi, Z.: VHDL: Analysis and modeling of digital systems. McGraw-Hill, Inc. (1997)

[104] Nipkow, T.: Hoare logics in isabelle/hol. In: Proof and System-Reliability, pp. 341–367. Springer-Verlag (2002)

[105] Olderog, E.R., Dierks, H.: Real-Time Systems - Formal Specification and Automatic Verification. Cambridge University Press (2008)

[106] Olderog, E.R., Swaminathan, M.: Layered composition for timed automata. In: Proceedings of the 8th international conference on Formal modeling and analysis of timed systems. pp. 228–242. FORMATS'10, Springer-Verlag, Berlin, Heidelberg (2010)

[107] Olderog, E.R., Swaminathan, M.: Structural transformations for data-enriched real-time systems. In: Johnsen, E., Petre, L. (eds.) Integrated Formal Methods, Lecture Notes in Computer Science, vol. 7940, pp. 378–393. Springer Berlin Heidelberg (2013)

[108] Paulson, L.C.: Isabelle: A generic theorem prover, vol. 828. Springer-Verlag (1994)

[109] Peter, H.J., Mattmuller, R.: Component-based abstraction refinement for timed controller synthesis. In: Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE. pp. 364–374. IEEE (2009)

[110] Peter, H.J.: A Uniform Approach to the Analysis and Complexity of Succinct Systems. Ph.D. thesis, Universität des Saarlandes (2012)

[111] Peter, H.J., Ehlers, R., Mattmüller, R.: Synthia: Verification and synthesis for timed automata. In: Computer Aided Verification. pp. 649–655. Springer-Verlag (2011)

[112] Pettersson, P.: Modelling and verification of real-time systems using timed automata: theory and practice. Ph.D. thesis, Citeseer (1999)

[113] Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57. IEEE (1977)

[114] Podelski, A., Wies, T.: Counterexample-guided focus. In: Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. pp. 249–260. POPL '10, ACM, New York, NY, USA (2010)

[115] Queille, J.P., Sifakis, J.: Specification and verification of concurrent systems in cesar. In: Proceedings of the 5th Colloquium on International Symposium on Programming. pp. 337–351. Springer-Verlag, London, UK, UK (1982)

[116] Rappaport, T.S., et al.: Wireless communications: principles and practice, vol. 2. Prentice Hall PTR New Jersey (1996)

[117] Rémy, D.: Using, understanding, and unraveling the ocaml language from practice to theory and vice versa. In: Applied Semantics, pp. 413–536. Springer-Verlag (2002)

[118] Rensink, A., Wehrheim, H.: Weak sequential composition in process algebras. In: Jonsson, B., Parrow, J. (eds.) CONCUR '94: Concurrency Theory, Lecture Notes in Computer Science, vol. 836, pp. 226–241. Springer Berlin Heidelberg (1994)

[119] Salah, R., Bozga, M., Maler, O.: On interleaving in timed automata. In: Baier, C., Hermanns, H. (eds.) CONCUR 2006 – Concurrency Theory, Lecture Notes in Computer Science, vol. 4137, pp. 465–476. Springer Berlin Heidelberg (2006)

[120] Saïdi, H., Shankar, N.: Abstract and model check while you prove. In: Halbwachs, N., Peled, D. (eds.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 1633, pp. 443–454. Springer Berlin Heidelberg (1999)

[121] Scheidler, C., Heiner, G., Sasse, R., Fuchs, E., Kopetz, H., Temple, C.: Time-triggered architecture (tta). Advances in Information Technologies: The Business Challenge pp. 758–765 (1997)

[122] Sipser, M.: Introduction to the Theory of Computation. Thomson Course Technology, 2nd edn. (2006)

[123] Sofronie-Stokkermans, V.: Hierarchic reasoning in local theory extensions. In: CADE. pp. 219–234 (2005)

[124] Sofronie-Stokkermans, V.: Locality results for certain extensions of theories with bridging functions. In: CADE (2009)

[125] Sorea, M.: Verification of real-time systems through lazy approximations. Ph.D. thesis, University of Ulm, Germany (2004)

[126] Thomas, D.E., Moorby, P.R.: The Verilog hardware description language, vol. 2. Springer-Verlag (2002)

[127] Torvalds, L.: The linux edge. Communications of the ACM 42(4), 38–39 (1999)

[128] Tripakis, S.: Checking timed büchi automata emptiness efficiently. In: Formal Methods in System Design. pp. 267–292 (2005)

[129] Valmari, A.: The state explosion problem. In: Lectures on Petri nets I: Basic models, pp. 429–528. Springer-Verlag (1998)

[130] Van Benthem, J., Doets, K.: Higher-order logic. In: Handbook of philosophical logic, pp. 275–329. Springer-Verlag (1983)

[131] Wachter, B., Westphal, B.: The spotlight principle. In: VMCAI. pp. 182–198 (2007)

[132] Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobald, C., Topić, D.: Spass version 2.0. In: Automated Deduction—CADE-18, pp. 275–279. Springer-Verlag (2002)

[133] Whittaker, J.: What is software testing? and why is it so hard? Software, IEEE 17(1), 70–79 (Jan 2000)

[134] Wies, T.: Symbolic Shape Analysis. Ph.D. thesis, University of Freiburg (2009)

[135] Wies, T., Kuncak, V., Lam, P., Podelski, A., Rinard, M.: Field constraint analysis. In: Proc. Int. Conf. Verification, Model Checking, and Abstract Interpratation (2006)

[136] Wies, T., Muñiz, M., Kuncak, V.: Deciding functional lists with sublist sets. In: Proceedings of the 4th International Conference on Verified Software: Theories, Tools, Experiments. pp. 66–81. VSTTE'12, Springer-Verlag, Berlin, Heidelberg (2012)

[137] Wies, T., Muñiz, M., Kuncak, V.: An efficient decision procedure for imperative tree data structures. Automated Deduction–CADE-23 pp. 476–491 (2011)

[138] Yannakakis, M., Lee, D.: An efficient algorithm for minimizing real-time transition systems. In: Courcoubetis, C. (ed.) Computer Aided Verification, Lecture Notes in Computer Science, vol. 697, pp. 210–224. Springer Springer-Verlag (1993)

[139] Yoneda, T., Shibayama, A., Schlingloff, B.H., Clarke, E.M.: Efficient verification of parallel real-time systems. In: Computer Aided Verification. pp. 321–332. Springer-Verlag (1993)

[140] Yovine, S.: Kronos: A verification tool for real-time systems. International Journal on Software Tools for Technology Transfer (STTT) 1(1), 123–133 (1997)

[141] Yovine, S.: Model checking timed automata. In: Rozenberg, G., Vaandrager, F. (eds.) Lectures on Embedded Systems, Lecture Notes in Computer Science, vol. 1494, pp. 114–152. Springer Berlin Heidelberg (1998)

[142] Zee, K., Kuncak, V., Rinard, M.: Full functional verification of linked data structures. In: Proceedings of the 2008 ACM SIGPLAN conference on Programming language design and implementation. pp. 349–361. PLDI '08, ACM, New York, NY, USA (2008)

Index